Automated PCB Component Placement using Reinforcement Learning

Luke Vassallo

Supervisor: Dr Josef Bajada

September, 2023

Submitted in partial fulfilment of the requirements for the degree of M.Sc. Artificial Intelligence.

L-Università ta' Malta
Faculty of Information &
Communication Technology

L-Università ta' Malta

www.um.edu.mt

Second edition, Thursday 7th September, 2023

All the wonders of our universe can in effect be captured by simple rules, yet there can be no way to know all the consequences of these rules, except in effect just to watch and see how they unfold.

(Stephen Wolfram)

Acknowledgements

I wish to convey my heartfelt appreciation to everyone who contributed to the realisation of this thesis and played a substantial role in shaping this remarkable phase of my journey.

First and foremost, I am immensely grateful to Dr Josef Bajada, my M.Sc. advisor, whose guidance and mentorship have been invaluable to me. He introduced me to reinforcement learning and the world of research, which have profoundly influenced my professional development. His extensive expertise in planning and reinforcement learning, coupled with his exceptional leadership, played a crucial role in delivering the contributions of this work. I could not have asked for a better mentor to assist me.

I would like to thank Dr Vincent Vella, Dr Jean Paul Ebejer and Prof. Luca Iocchi for examining my thesis and Prof. Matthew Montebello for chairing the process. Their insightful questions and detailed comments have significantly improved the quality of this thesis.

Next, I would like to thank Marja for her boundless patience, unwavering encouragement and dedication. I would also like to extend my thanks to my parents, Emanuel and Ruth, my brother Darren and uncle Geoffrey, for encouraging me to follow my passions and supporting me in pursuit of my goals with their unconditional love.

Lastly, I want to thank everyone else who shared an ear or provided an opinion, whether directly or indirectly related to this thesis. These conversations have been a source of inspiration, and some of their ideas live through this work.

I am grateful to the Malta Digital Innovation Authority (MDIA) for their support through the Pathfinder scholarship.

Abstract

Component placement is the first step of Printed Circuit Board (PCB) physical design and demands considerable time and domain expertise. Placement quality impacts the performance of subsequent tasks, and the generation of an optimal placement is known to be, at the very least, NP-Complete. While stochastic optimisation and analytic techniques have had some success, they often lack the intuitive understanding of human engineers. Consequently, automation tools have not gained acceptance among hardware engineers, resulting in predominantly manual PCB physical design. Moreover, limited literature on this topic is available, with a particular focus on placement co-optimisation for niche applications. We aim to challenge this prevailing approach and propose a general placement methodology that integrates an AI-assisted workflow, shifting the designer's focus towards higher-level problem-oriented tasks.

This thesis investigates state-of-the-art AI constructive placement methodology and, based on its shortcomings, proposes a novel Markov Decision Process (MDP) formulation for iterative placement. We study the fundamental mechanisms in a constrained single-component setup using advanced policy optimisation and actor-critic Reinforcement Learning (RL) algorithms. Furthermore, we evaluate the feasibility of encouraging the agent to learn by emulating the design style of experts or by acting as an expert itself in an iterative self-improvement scenario. Based on the key findings, we adapt it to a multi-component setup and learn general policies for optimising the placement of components on unseen PCBs. Our research suggests that an adaptive reward signal removes the need for expert knowledge, and the multi-component environment yields diverse training datasets sufficient for learning general policies from a handful of circuits. Our methodology is evaluated on unseen circuits alongside simulated annealing, an effective methodology for optimising the placement of circuit components on a PCB. TD3 and SAC, on average yield 17% and 21% lower post-routing wirelength. Qualitative analysis shows that the policies learn fundamental placement techniques. Collectively, they demonstrate emergent collaborative and competitive behaviours between components and faster placement convergence over an order of magnitude. The methodologies have been open sourced and are available on GitHub `https://github.com/lukevassallo/rl_pcb.git`

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

1 Introduction

The PCB physical design process involves two steps: placement and routing. Placement determines the precise geometrical positions of all the components in the circuit, while routing involves drawing wires to perform connections as described by the schematic (Kaeslin, 2008, p .40). This thesis focuses on the placement step which places all the circuit components in the netlist onto a pre-defined layout region while minimising circuit parameters such as wirelength and component overlap. Automated tools often formulate the problem as a multi-objective optimisation, minimising a combination of these two parameters (Kahng et al., 2022, p .95). At a high-level, component placers are classified as constructive or iterative. The former order the circuit netlist according to a criterion such as component size or connection density and sequentially place them onto an empty layout region. The latter starts with random placement and is iteratively improved until a terminal condition is reached or a pre-defined amount of steps have been performed.

1.1 Motivation

Hardware design offers an extra degree of freedom for product differentiation. However, small design teams may lack intricate knowledge of the subject and find it challenging to tackle such an endeavour. By contrast, those with in-house expertise spend significant time performing physical design. From a product perspective, system-level design, selection and interconnectedness of components are often of the most value. Designing the PCB that meets the requirements of the product is often seen as a transformation of the abstract setup that does not augment these features. However, it requires technical expertise and knowledge of physical principles to perform this task reliably, meeting requirements and conforming to standard certifications. The duration of the design process depends on various factors ranging from design complexity to resources to the team's expertise and, as a result, may span from days to months. By abstracting complexity through acPCB design automation, we enable another design dimension to enhance product differentiation in the marketplace and accelerate innovation through custom solutions.

The Electronic Design Automation (EDA) field encompasses design flows for PCBs and Integrated Circuits (ICs) amongst others under which physical design falls. Like many physical design processes, placement is at least Nondeterministic Polynomial time - Complete (NPC) (Garey et al., 1976), meaning that a placement solution can neither be found nor verified in polynomial time. The PCB and IC design processes have much in common, but the scale is one of the fundamental ways they differ.

The EDA field encompasses design processes for PCBs and ICs, with placement being a key physical design task. Placement is at least NPC (Garey et al., 1976), meaning that a solution can neither be found nor verified in polynomial time. Although PCB and IC designs share similarities, their scale is a distinguishing factor. Khailany et al. (2020) notes that the work done per IC has increased while the design time has remained roughly the same. This indicates that manual effort has been offloaded to software automation tools, which have become crucial for taping ICs. As a result, IC physical design is an actively researched topic in academia (Gao et al., 2023; Lin et al., 2021; Lopera et al., 2021; Markov et al., 2015) and industry (Agnesina et al., 2023; Ho et al., 2023a; Khailany et al., 2020). By contrast, PCB layout is still tractable to manual layout, so it is still predominantly performed by hand. For this reason, in recent years, little research has been published on automating the physical design, with some notable exceptions that treat specific challenges (Alexandridis et al., 2017; Ning et al., 2020). Throughout this thesis, we look at both fields to draw inspiration for proposing our methodologies.

Deep learning is seeing great potential in EDA for reducing design time by replacing time-consuming processes with accurate predictions (Xie et al., 2021; Zhang et al., 2020) and accelerating the time to solution by reducing iterative development cycles through expert predictions (Agnesina et al., 2020). Due to the NPC nature of the problem in physical design, current state-of-the-art use non-linear optimisation (Cheng et al., 2019; Lin et al., 2015, 2021), which require formulating a differentiable cost function. That latter becomes increasingly difficult to define when the number of objectives and constraints increases. When metaheuristics are utilised (Sechen and Sangiovanni-Vincentelli, 1985), this is generally easier; however, these methods have trouble scaling (Cohoon and Paris, 1987; Markov et al., 2015) to large problem sizes, which is less concerning for PCB than IC design. Recently deep RL has been applied to automate specific physical design steps in part (Huang et al., 2019; Xu et al., 2022) or in an end-to-end manner (Mirhoseini et al., 2021), albeit their contributions are still in their infancy. Deep RL offers attractive solutions to such problems, particularly its ability to represent vast state spaces - which these problems are notorious for - and leverage experience for solving tasks. Recent applications in deep learning for PCB design have shown promise (Liao et al., 2020; Murphy, 2020), but the ones using RL (Crocker, 2021) still have plenty of room for improvement.

1.2 Aims and Objectives

Our goal is to create an Artificial Intelligence (AI) driven component placement engine capable of generating high-quality solutions on unseen circuits. We break down the goal into the following four objectives.

1. To assess the performance of a Reinforcement Learning (RL) constructive PCB placer by developing a circuit quality estimator and using it as the policy state encoder.
2. Formulate the iterative PCB component placement problem as an RL task.
3. To systematically design and evaluate the fundamental mechanisms of a single-component iterative PCB placement environment, namely, action spaces, reward signals and training algorithms.
4. To design and evaluate a multi-component RL setup for iterative PCB component placement capable of learning generalisable behaviour.

1.3 Proposed Solution

This thesis focuses on methodologies for optimising the placement of components onto a PCB using RL. We investigate the state-of-the-art RL placement methodologies (Mirhoseini et al., 2021; Xu et al., 2022) in literature and propose a novel MDP formulation for iterative placement. In our formulation, each component is treated as an autonomous system capable of perceiving its surroundings, and with knowledge of the target aims to learn policies that optimally co-locate it within a circuit. We investigate the mechanics of our formulation in constrained single-component environments using policy optimisation and actor-critic methods. These findings are then adapted to a multi-component setup, allowing for diverse and consistent data collection. Under the guidance of an adaptive reward signal, we attempt to learn general policies from scratch without expert knowledge, capable of optimising circuits not encountered during training. The resulting methodology is evaluated regarding post-routing wirelength on unseen circuits alongside an open vanilla implementation of Simulated Annealing (SA) (Holtz et al., 2020; Merrill, 2021). The latter has proven highly effective for placing circuits with typical component sizes in PCBs (Cohoon et al., 1991; Kirkpatrick et al., 1983) and has achieved notable commercial success (Sechen and Sangiovanni-Vincentelli, 1985). Our results suggest that leveraging experience yields a higher quality layouts with up to 21% lower post-routing wirelength. In studying this problem, we collect a set of 30 circuits from real-world applications due to the absence of a public dataset suitable for PCB physical design.

1.4 Document Structure

This thesis is divided into six chapters, including this introductory chapter.

Chapter 2 describes elementary concepts from Electronic Design Automation (EDA), Machine Learning (ML) and Reinforcement Learning (RL), all of which are fundamental to understanding the work present in this thesis.

Chapter 3 surveys seminal works in PCB and IC physical design culminating in the application of ML to EDA processes. We identify the state-of-the-art while showing the revolutionary potential of this technology within EDA.

Chapter 4 contains our methodology split into three sections. Objectives two and three are tackled simultaneously in this chapter's second section. We document methods and describe the experimental setups of all tests carried out.

Chapter 5 follows an identical flow and presents the experimental results. These are accompanied by an evaluation against Simulated Annealing (SA) and an analysis of learned behaviour. The following discussion provides an ML and EDA perspective of the work.

Chapter 6 concludes this thesis by outlining our contributions, limitations of our work and potential avenues for future research.

Appendix A thoroughly describes the selection of toolchains, modifications performed and experimental setup considerations.

Appendix B provides a complete description of the datasets used in this thesis. For randomly generated datasets, the settings are provided.

Appendix C provide supplementary experiments to understand parameter relationships and make data-driven decisions.

Appendix D contains additional background information.

Appendix E contains machine and detailed experiment timing information.

2 Background

PCB component placement is at least an NPC problem (Garey et al., 1976) and an optimal solution NPC problem can neither be attained in polynomial time nor verified because no known method exists. The placement task accepts a circuit netlist describing the logical representation of the circuit accompanied by geometrical information of the individual components (e.g. width and height) (Kaeslin, 2008, p. 40). The placement process is concerned with identifying the best spatial location and orientation for all the components given solution constraints (e.g. no overlaps) and optimisation goals (e.g. minimised wirelength) (Kahng et al., 2022, p. 95). Since we cannot verify a solution, a cost function that combines overlap and wirelength is often defined and minimised.

This section presents background information that helps the reader understand the work and contributions of this thesis. It introduces the field of EDA, specifically focusing on physical design, and highlights the similarities between PCB and IC design processes. Since the circuit netlist naturally exhibits graph representation, we introduce Graph Neural Networks (GNNs) as automatic feature extraction methods. Lastly, it offers a concise overview of RL, the method of choice for solving placement-related combinatorial optimisation tasks, covering basic concepts to advanced algorithms.

2.1 Electronic Design Automation

The Electronic Design Automation (EDA) industry develops software to support engineers in creating new electronic designs. Due to the high complexity of modern designs, EDA touches almost every aspect of the design flow, from specification to fabrication. EDA addresses designers' needs at multiple levels of the electronic system hierarchy, including ICs, Field Programmable Gate Arrays (FPGAs), Multi Chip Modules (MCMs), and PCBs.

ICs comprise many miniature electronic components, primarily transistors, built into a monolithic semiconductor substrate via a photolithographic process. Many circuits are manufactured onto a circular silicon wafer before slicing into individual bare dies ranging from the size of a pinhead to a large postage stamp. Dies may be encapsulated into a

plastic or ceramic package before being soldered onto a PCB (Kaeslin, 2008). MCMs are a collection of ICs and other components aggregated onto a single substrate while FPGAs are specialised programmable ICs that can implement arbitrary logic circuits.

2.2 IC Design Process

Very Large Scale Integration (VLSI) circuits are highly complex. The design process is separated into distinct steps as outlined by the flowchart to the left in Figure 2.1. The initial phases contribute towards the overall system functionality and logical design and are carried out at a high level of abstraction. Later steps take place at a lower level of abstraction, where software tools use geometrical information about the constituent devices to layout the circuits and perform interconnection between transistors and modules. Finally, the focus shifts towards verifying correct logical operation, deriving electrical properties and analysing their effect on performance (speed, power), reliability and manufacturing yield.

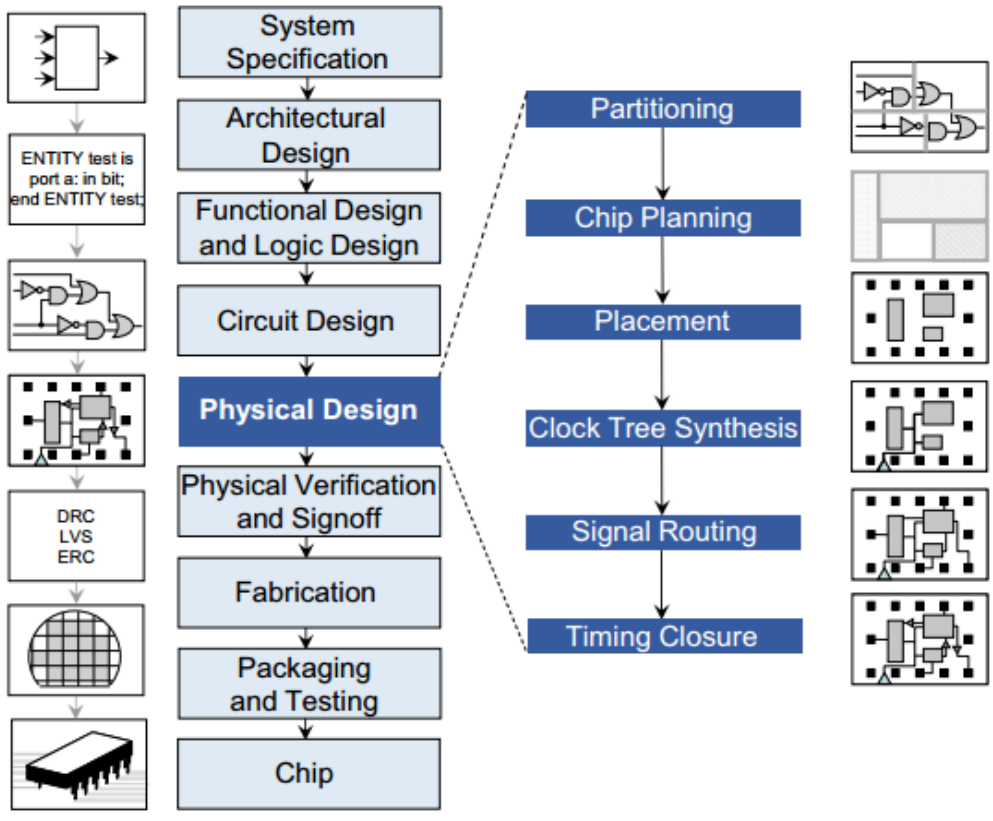


Figure 2.1: Key steps in VLSI circuit design flow with an emphasis on physical design (Kahng et al., 2022, p. 6).

On the left of Figure 2.1, the steps of the VLSI design flow are illustrated with a focus on the physical design on the right. The IC physical design process inspires most of the work carried out in this dissertation.

2.2.1 Design Flow

The design flow is a set of steps that allows a team of designers to progress from specifications to a fabricated IC. The product specifications are defined by bringing together designers, product marketers, and managers to establish high-level requirements of the product in terms of functionality, manufacturing and performance. These goals are converted to architectural specifications that outline the main system components and how they will be brought together. The engineering team then performs functional and logic design and uses test benches to verify the correct operation. The following circuit design phase consists of synthesis that converts the high-level functional description (e.g. Verilog) into basic circuit elements (logic gates and transistors) described by a netlist. The beginning of physical design is marked by the instantiation of all design elements in their geometric representation from the synthesised netlists. Partitioning and floorplanning place large clusters of transistors and Intellectual Property (IP) subsystems while optimising for wirelength, routability and other performance metrics. Placement fixes the size of all macros and standard cells and assigns each to a spatial location on the chip. Routing connections are made using metal layers typically running above the standard cells. The physical design process generates a set of layouts and manufacturing specifications that require verification. Following verification, the chip is said to tape out and is sent for fabrication. The fabrication process uses circular wafers from pure silicon and photolithographic masks to etch successive patterns onto a silicon wafer. Chemical processes follow to gradually build the chips layer upon layer. After fabrication, each chip on the wafer is individually tested, and the functional ones are packaged for placement onto a PCB.

So far, we have presented an overview of the IC design process. The upcoming sections describe the individual steps of the physical design process as illustrated by the right flowchart in Figure 2.1. A simplified view of the objective is to minimise the total wirelength since this directly correlates to Power Performance and Area (PPA). It is a crucial step as it impacts circuit performance, area, reliability, power, and manufacturing yield.

2.2.1.1 Partitioning and Floorplanning

The synthesised netlist serves as the circuit description for physical design. It may optionally be partitioned if the netlist is large or comprises macros. This is done by clustering groups of transistors or standard cells into soft cells (clusters of defined area and variable

aspect ratio). The soft and hard cells are used in the next step of floorplanning. Floorplanning is responsible for laying out all the macros and clusters of standard cells in a way that minimises area and maximises performance. The latter is quantified by estimation of wirelength and results in a placement that reduces connection length between modules. The floorplanning step identifies all parameters pertaining to cell dimensions and legal assignment on the chip canvas. The quality of a floorplan can significantly influence the performance of subsequent steps.

2.2.1.2 Placement

Placement is the process of placing each cell and macro onto the chip canvas while having some performance objectives related to PPA and constraints such as reduced overlap or an upper bound on a specific parameter. Historically placement algorithms can be categorised into four distinct sections, partition based, meta-heuristic (black-box optimisation), mixed and analytical. Placers are discussed thoroughly in the section 3.2. Placement is typically divided into two stages: global and detailed. Global placement involves finding a rough placement for all the transistors and macros. Since the result of global placement may contain overlap, in detailed placement, the placement of the elements is altered slightly to reduce the wirelength further and obtain a layout without any overlap.

2.2.1.3 Routing

Routing takes place after placement and determines the paths connecting all the circuit elements. This process has some objectives, such as achieving 100% routability and wiring all points in the circuit. Other objectives may include reducing the overall wirelength and achieving a certain wirelength for critical nets to satisfy the timing budget. The latter is vital in modern circuits since a worse timing budget leads to either incorrect functioning of the IC or derating to a lower speed grade. Routing is split into distinct steps starting with the clock network and followed by a two-stage signal routing, global and detailed.

In digital circuits the clock signal is considered the most critical signal. It is often assigned its routing network so it can be separately optimised and has less influence from logic signals. Routing clock signals and circuits placed in its path (e.g. gated buffers) is prioritised and performed prior to routing the remainder of the circuits.

Global routing partitions the placed layout into tiles and allocates nets to tile-to-tile paths. Routing capacity is associated with each path, and nets are assigned without exceeding the capacity. Critical nets may be weighted such that they have a higher contribution in terms of capacity. Detailed routing is responsible for making the individual connecting traces and is considered the most challenging part (Khailany et al., 2020). Static

timing analysis is run, including the routing loads placed on the logic gates. Depending on the result, the design may iterate multiple times from logic synthesis, placement and routing. Subsequent synthesis and placement runs can be optimised for timing closure, where critical paths are prioritised to minimise wire delay and meet the performance goals.

2.3 PCB Design Process

A PCB provides the structure for mounting and interconnecting circuit components (IPC, 2012). This section describes different categories of circuit components, the PCB structure and associated design flow.

2.3.1 Components

Components are the constituent devices that make up an electronic circuit and fall into three categories, electronic, electromechanical, and mechanical components. *Electronic components* influence the flow of electrons and can be further classified as passive (e.g. resistors, capacitors and inductors) and active (e.g. semi-conductor devices). *Electromechanical components* involve a combination of electrical and mechanical effects. An electrical current may promote or inhibit mechanical movement, and vice versa, whereas a mechanical action may promote or inhibit current flow. Examples include quartz crystals and motors. *Mechanical components* do not explicitly influence an electrical signal and include connectors and wires.

2.3.1.1 Component Packages

From a PCB layout perspective, circuit components can be categorised based on their package type, either through-hole or Surface Mount Devices (SMDs). Through hole components have rigid leads inserted into drilled holes on the PCB and are highly reliable due to a robust mechanical connection (Preston, 2018). On the other hand, Surface mount devices have flat co-planar leads, and the component lies flat on the PCB. A mechanical connection is created between the component's lead and the exposed pad on the board's surface. Since no holes are required, high-density PCBs are achievable both due to SMD devices being smaller and since no drilling is involved, components can be laid on both sides of the board. SMD devices have been widely adopted due to their lower cost and smaller size and are generally more amenable to automated processes. Figure 2.2 illustrates double-sided component placement and depicts various packages adopted by the industry optimised for different applications such as density, cost and ease of soldering.

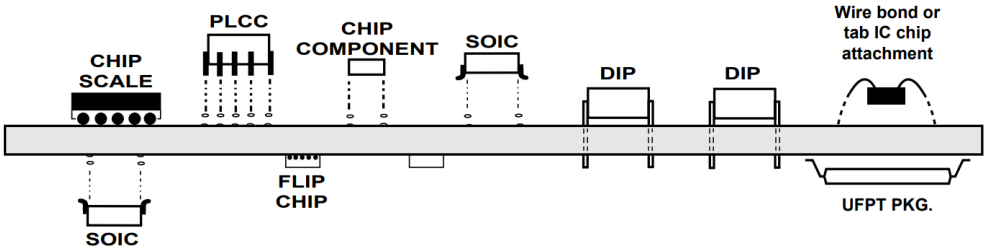


Figure 2.2: Comparison of through-hole and surface mount package technologies (IPC, 2012).

2.3.1.2 Symbols and Footprints

Every component has a schematic symbol and an associated footprint. The schematic symbol defines the function of the components and communicates its interface to the user and the Computer Aided Design (CAD) tool. The footprint defines the geometrical representation of the component in the appropriate physical dimensions, orientation and outline of the package. In other words, symbols define the connectivity of a component, and the footprints map this connectivity into geometrical patterns corresponding to the physical dimensions of the actual component. Institute for Interconnecting and Packaging Electronic Circuits (IPC) and American National Standards Institute (ANSI) standardise schematic symbols and their footprints. However, the designer may also create custom symbols and footprints that do not follow a standard.

2.3.2 PCB Structure

A PCB is a three-dimensional structure with alternating copper and insulating layers. Traces (wires) are etched on copper layers, and connect component pins to form a closed circuit for current to flow. Wires may propagate vertically across layers using a Vertical Interconnect Access (VIA). Figure 2.3 illustrates a cross-section of a multi-layer PCB and highlights the key elements. The top and bottom copper layers contain both component pads and interconnecting traces (see Figure 2.4(b)) for routing signals and power. Internal layers are strictly limited to wiring. In the simplest form, a PCB may have a single copper layer. However, due to low cost, two layers are often used as a minimum. Advanced PCBs contain many internal copper layers to increase the routing capacity for high-density layouts. The key elements found on a PCB are:

- *Pads* are geometrical shapes etched onto the PCB's top and bottom layers. They

provide the regions onto which components pins are soldered.

- *Traces* are connections etched on the top, bottom and inner copper layers of a PCB. They act as wires providing a connection between pads.
- *VIA* are vertical holes in the PCB that provide an electrical connection between layers. That is, a trace on the top layer can bridge to another layer through a VIA.

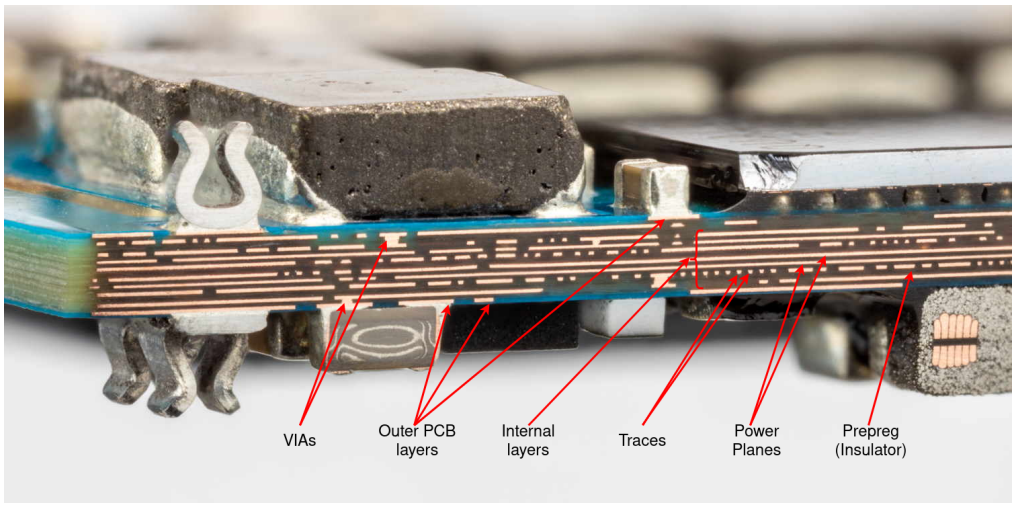
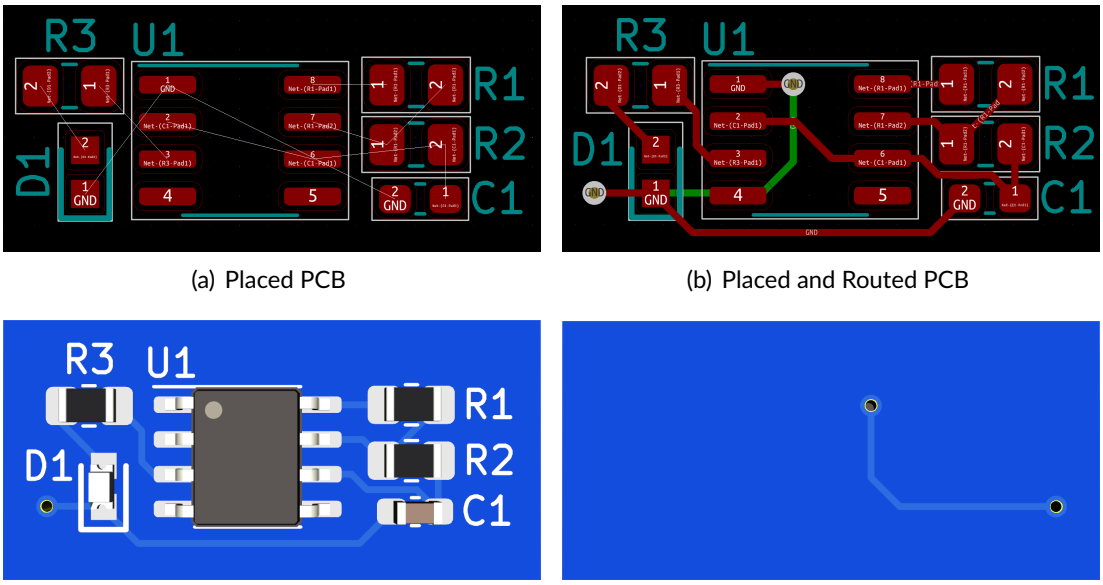


Figure 2.3: Cross-section of a Printed Circuit Board (PCB) (Oskay and Schlaepfer, 2022, p. 16).

Figure 2.4 illustrates a PCB in KiCad (Bautista et al., 2022) software. Figure 2.4(a) depicts the placed circuit in the layout tool. The white lines make up the *ratsnest* which communicates the pins that need to be connected with traces. Figure 2.4(b) shows the routed circuit, with red and green traces being wires on the top and bottom layers, respectively. Notice further that the VIA connects both layers. Figures 2.4(c) and 2.4(d) depicts a three-dimensional render of the PCB respectively from the top and bottom.

2.3.3 Design Flow

The design of an electronic system can be divided into five stages: specifications, schematic capture, physical design, manufacturing and assembly, and testing. A project begins with high-level requirements generated by understanding the product's market fit, customers' needs, and technological possibilities and capabilities. The resulting requirements may include features, certifications, mechanical or power constraints, and cost considerations. Based on these high-level requirements, engineers design electronic circuits that fulfil the



(a) Placed PCB

(b) Placed and Routed PCB

(c) Top view of 3D rendered PCB

(d) Bottom view of 3D rendered PCB

Figure 2.4: Illustration of placed and routed circuits alongside their corresponding 3D render in the KiCad (Bautista et al., 2022) design software.

design criteria. This process involves identifying the necessary components, considering availability, price, and performance, and creating a schematic. The schematic is a graphical representation of the electronic design, displaying component symbols and their interconnections. It communicates the design visually, and a netlist can be automatically generated. The circuit can then be simulated or prototyped to verify that it operates as intended. These tasks minimise the risk of developing a non-functional electronic system.

PCB physical design starts by importing the netlist and arbitrarily placing all the component footprints on the board. There is no standard way to perform the layout process, and every engineer tends to do it differently. However, it is generally understood that good component placement is crucial for routability and overall performance (Jones and Harris, 2004). For advanced PCBs, the performance of an electronic system is highly dependent on the PCB layout and, if not done properly, may result in a system that fails compliance testing or exhibit non-functional sections. After the PCB has been placed and routed, a Design Rule Check (DRC) checks for layout errors and manufacturing constraints. The manufacturing process of PCB involves a series of lithographic and chemical procedures and is rarely done in-house. Many PCB manufacturers offer assembly services where they solder either all or a portion of the components. Alternatively, engineers may also assemble prototype units manually. The final stage involves testing, which aims to identify bugs and verify that the desired performance has been achieved.

2.4 Graph Neural Networks

The language of graph structures elegantly captures numerous natural and artificial patterns Veličković (2023). This is especially true for circuit netlists in EDA as graphs naturally capture structural relationships and dependencies. This section introduces GNNs that operate on non-Euclidean graph data and will be utilised in the thesis to extract low-level features directly from circuit netlists.

A graph denoted by $G = (V, E)$ consists of a set of nodes (vertices) V and a set of edges E. Directed edges are comprised of pairs of nodes in the form of (v, w), which indicate that an edge connects node v to w. Undirected edges are denoted by $\{v, w\}$.

2.4.1 Graph Neural Networks

Many authors emphasise the success of automatic feature extraction (Gilmer et al., 2017; Wu et al., 2021), concluding that better representations are obtained over handcrafted feature engineering and in significantly less time. Wu et al. (2021)'s comprehensive survey notes that GNNs, compared to other feature extraction methods such as Convolutional Neural Networks (CNNs), have three challenges:

1. Graph data is irregular, having a variable number of nodes, each with a different amount of connections to its neighbours.

2. ML algorithms assume data instances are independent of each other, which does not hold for graph data because nodes are related to their neighbours via edges.

3. Due to the isomorphic properties of graphs, methods that operate on graph data need to be permutation invariant.

GNNs are categorised into four classes (Wu et al., 2021). During their inception, recurrent neural networks were used to recursively update the node features within a graph as a function of their neighbours until an equilibrium state was achieved (Gori et al., 2005; Sperduti and Starita, 1997). Graph Convolution Networks (GCNs), on the other hand, introduced a message-passing mechanism inspired by CNNs, which executes localised operations without recurrent connections (Bronstein et al., 2017). These GCNs can be further divided into spectral approaches (Bruna et al., 2013) and spatial approaches (Micheli, 2009). Building upon the concept of message passing, graph autoencoders were developed (Kipf and Welling, 2016; Pan et al., 2018). These models convert the graph into an intermediate representation and then reconstruct it, aiming to learn unsupervised repre-

sentations. Lastly, spatial-temporal GNNs are designed to handle data with both spatial and temporal dimensions.

The upcoming sub-section presents the Message Passing Neural Network (MPNN) (Gilmer et al., 2017), a general framework that fits many proposed ML layers that operate graph data. Subsequent sub-sections briefly describe graph convolution networks and masked attention later used for learning compressed representations of circuit netlists.

2.4.1.1 Message Passing Layer

Gilmer et al. (2017) proposed the MPNN as a general framework for supervised learning directly from graph data in a permutation-invariant manner. The framework is designed to handle models that operate on uni-directed graphs and consists of two stages: an aggregation step and an update step. In the aggregation step, neighbour nodes transmit messages along graph edges, optionally conditioning the node data. The update step utilises the resulting embedding of the neighbour nodes to update the central node.

Formally, the MPNN layer accepts a graph G with node and edge attributes x_v and e_{vw} respectively. A typical forward pass is comprised of T time steps, each performing an individual aggregate and update, respectively represented by the message function, M_t and vertex updated function U_t.

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.1}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.2}$$

Equation 2.1 illustrates the aggregate phase, where messages m_v^{t+1} are computed as a function of the node v, it's neighbours $N(v)$, and attributes, e_{vw} linking them. Equation 2.2 describes the update phase where the hidden states h_v^t, and current node v are updated based on the messages m_v^{t+1}. Equation 2.3 captures an optional readout phase, which collapses the whole graph onto a single feature vector using a readout function R.

$$y = R(h_v^T | v \in G) \tag{2.3}$$

The MPNN framework enables performing classification or prediction tasks on individual nodes and testing for the existence of links between nodes. Additionally, when a separate readout function is defined, the graph is collapsed on a single feature vector, which can subsequently be used for graph-level prediction tasks.

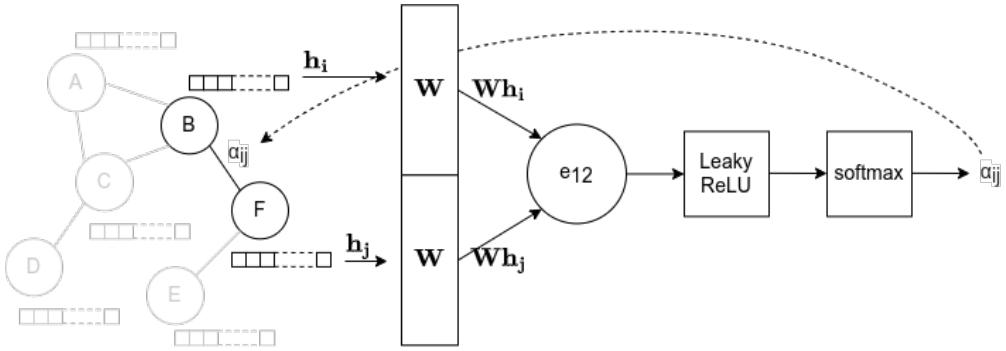


Figure 2.5: The architecture of a GAT layer showing key components and computation of the attention between two nodes.

2.4.1.2 Graph Convolutional Networks

As mentioned earlier, GCNs can be classified into spectral and spatial-based approaches. Spectral approaches build upon the seminal work of Bruna et al. (2013) and perform convolutions based on spectral graph theory or its derived approximations. Subsequent studies by Henaff et al. (2015), Defferrard et al. (2016), and Kipf and Welling (2017) have further extended this concept. In contrast, spatial methods draw inspiration from traditional CNNs and define graph convolutions through information propagation. Spatial GNNs were initially introduced by Micheli (2009), and despite not achieving immediate success, subsequent works by Atwood and Towsley (2016), Niepert et al. (2016), and Gilmer et al. (2017) have reignited interest in spatial GNNs.

2.4.1.3 Attention Mechanism in GNNs

Figure 2.5 illustrates the topology of a Graph Attention Network (GAT), which augments a spatial GNN with masked self-attention (Veličković et al., 2017). Equation 2.4 mathematically captures the computation of the attention coefficient where α_{ij} is the normalised attention coefficient between nodes i and j, and W is a learnable weight that transforms the original node embeddings h_i and h_j to Wh_i and Wh_j. The attention co-efficient is yielded after propagating through a Leaky ReLU and softmax activations. Notice that the original embeddings h_i and h_j remain unchanged in this operation.

$$\alpha_{ij} = \frac{e^{LeakyReLU(W_a[Wh_i|Wh_j])}}{\sum_{k \in N(i)} e^{LeakyReLU(W_a[Wh_i|Wh_k])}} \tag{2.4}$$

Equation 2.5 shows the convolution operation performed to update the node attributes as a function of its neighbours and the attention coefficient, where, h'_i and h'_j are the updated node embeddings as a result of transforming them through a learnable

weight W scaled by the attention coefficient α_{ij}. This is a weighted version of a vanilla spatial GNN originally proposed by Micheli (2009).

$$h_i^{'} = \sigma(\sum_{j \in N(i)} \alpha_{ij} W * h_j) \tag{2.5}$$

2.5 Reinforcement Learning

RL is a type of machine learning where an agent learns to interact with an environment to maximise cumulative rewards. It is a learning paradigm inspired by how humans and animals learn through trial and error by receiving feedback from their surroundings. Unlike supervised learning, which involves learning tasks from expert-labelled data, or unsupervised learning, which focuses on identifying structures or clusters within historically unlabelled data, RL stands out by learning through trial and error with the environment, guided by a reward signal.

This section describes the RL paradigm and introduces relevant terminology. We explain the fundamental mathematical formulations and describe traditional approaches. We also introduce Deep RL and contrast advanced algorithms employed in the thesis for finding good policies that optimise the placement of components on a PCB.

2.5.1 Basics

In RL, the agent is an entity that learns by taking actions in the environment. The latter represents the external context in which the agent operates and is typically defined by a set of states and rules that govern the agent's actions. The agent interacts with the environment by observing its current state, selecting actions to alter it, and receiving feedback as a reward. The goal of RL is for the agent to learn an optimal policy—a strategy or set of rules—that guides its decision-making process. The policy determines the agent's actions in each state to maximise its expected long-term rewards.

Figure 2.6 illustrates the agent's interaction with the environment. At a given time t, the agent observes the environment and infers its state, S_t. Based on this information, it employs its policy π to select an action A_t. The action is then applied in the environment, and the episode advances to the next timestep $t+1$ where the agent observes the updated state S_{t+1} and is accompanied by a reward R_{t+1} reflecting the its effectiveness.

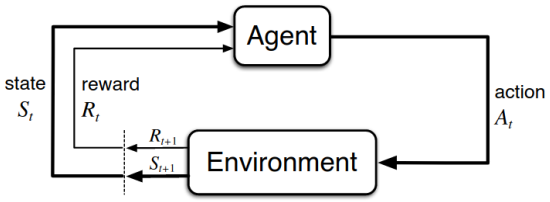


Figure 2.6: Agent-environment interaction in RL (Sutton and Barto, 2018, p. 48).

2.5.1.1 Terminology

This section provides essential definitions of terms used in problem formulation, algorithm description, and learning policies in the RL field.

- The *state*, S_t fully describes the environment's current condition. An observation, O_t, is what the agent perceives and may contain a subset of the state information. A fully observed state is one where the agent's observation encompasses the state entirely, and a partially observed state is one where the agent perceives a reduced version of the state and may require extra information to infer the state.
- The agent's set of valid actions is called the action space, and the agent's *action* sampled from it at timestep, t, is denoted by the random variable A_t. Action spaces can be classified as either discrete or continuous. A discrete action space comprises a finite set of possible actions, while a continuous action space involves real-valued vectors typically bounded by maximum and minimum values.
- The *reward* signal, R_t, provides feedback to the agent based on its actions, indicating the desired outcome and guiding it to learn optimal behaviour.
- An *episode* defines the timeframe for the agent's interaction with the environment. An RL task is classified as *episodic* if it has a terminal state and *continuous* otherwise.
- The *return* G_t, at timestep t is the expected sum of future rewards, starting from the current state s_t until reaching a terminal state, when following a policy, π. A distinction thus must be made between episodic and continuous episodes. Equation 2.6 illustrates how the return is computed for episodic tasks.

$$G_t = R_{t+1} + R_{t+2} + ...R_T \tag{2.6}$$

Continuous tasks do not have a terminal state; therefore, the expected return will result in an infinite value. To overcome this limitation, a discount factor is introduced, $\gamma \in (0, 1]$, which places more emphasis on upcoming rewards. Notice that if

gamma in Equation 2.7 has a value less than one, then the infinite sum has a finite value as long as the reward is bounded.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{2.7}$$

The value of γ is of particular importance. A value close to one tends to sum the return over a more extensive sequence of future rewards. In terms of the behaviour embodied by the policy, this will promote long-term decision-making. On the other hand, a smaller value of gamma will lead to short-sightedness, where the agent will seek actions that yield immediate reward.

2.5.1.2 Markov Decision Processes

An MDP is a formal probabilistic model (Sutton and Barto, 2018, p. 47) that represents sequential decision-making in RL. It captures how actions affect the current and future rewards and states. MDPs are an idealised version of the RL problem and formalise goal achievement through interaction. In a finite MDP, there is a finite set of states, actions, and rewards, and the random variables S_t and R_t have discrete probabilities. The dynamics of an MDP are precisely defined in Equation 2.8:

$$p(s', r|s, a) = Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \forall s', s \in S, r \in R, a \in A(s) \tag{2.8}$$

Where $p(s', r|s, a)$ is the probability of transitioning from the current state s taking action a and ending up in state s' receiving a reward r. Equation 2.9 shows that the probability of transitioning to state s' and gaining a reward r is conditional on the previous state s and action a.

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1 \tag{2.9}$$

2.5.2 The Bellman Equation

The Bellman equation (Sutton and Barto, 2018, p. 59) is a fundamental concept in RL that characterises the value of a state in terms of the expected future rewards. It provides a recursive relationship between the value of the current and successor states. The Bellman Equation is provided by Equation 2.10 where $V_\pi(s)$ represents the value of state s under policy π, indicating the expected cumulative reward, $\pi(a|s)$ represents the probability of

selecting action a in state s according to the policy π, $p(r, s'|s, a)$ is the state transition probability quantifying the likelihood of transitioning to state s' and receiving reward r for selecting action a in state s and γ is a discount factor.

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{r \in R} \sum_{s' \in S} p(r, s'|s, a)(r + \gamma V_\pi(s')) \tag{2.10}$$

2.5.3 Fundamentals RL Algorithms

RL algorithms can be categorised into two main types: model-based and model-free. Model-based RL relies on building an internal model of the environment to plan and make decisions, while model-free RL directly learns a policy or value function from interactions with the environment without explicitly modelling its dynamics. Model-free RL does not assume knowledge of the underlying model, thus making it suitable for highly complex tasks such as placement. Hence, this thesis focuses on model-free RL.

2.5.3.1 Value-based Methods and Policy Optimisation

Value-based methods focus on estimating the value function, which represents the expected cumulative reward from a particular state or state-action pair. These methods aim to learn the optimal value function by iteratively updating value estimates using techniques like Q-learning (Watkins and Dayan, 1992) or State Action Reward State Action (SARSA) (Sutton and Barto, 2018, p. 129). The learned value function guides decision-making by selecting actions with the highest value. Value-based methods are effective in high-dimensional or continuous state and action spaces but can suffer from convergence issues and can be sensitive to exploration strategies. On the other hand, policy optimisation methods aim to directly learn an optimal policy that maps states to actions without explicitly estimating value functions. These methods optimise the policy parameters using methods such as REINFORCE (Sutton et al., 1999). Policy optimisation methods can handle discrete and continuous action spaces and directly optimise complex policies. However, they may require a large number of samples for convergence and can be more computationally expensive than value-based methods.

2.5.3.2 On-policy and Off-policy

On-policy and off-policy are two distinct RL algorithm classes that differ in how they learn from interaction data. On-policy algorithms, such as SARSA (Sutton and Barto, 2018, p. 129), update their policy based on the experiences gathered by following the current

policy. In other words, the same policy being evaluated and improved is used for data collection. By contrast, off-policy algorithms such as Q-learning (Watkins and Dayan, 1992), learn from data collected by following a different behaviour policy than the one being improved. They use a separate target policy for evaluation and a behaviour policy for exploration. This decoupling of the target and behaviour policies allows for more efficient exploration and learning from a broader range of experiences. Off-policy algorithms have advantages in terms of data efficiency and stability since they can make use of historical data collected by any behaviour policy.

2.5.3.3 Exploration and Exploitation

Exploration-exploitation trade-offs play a crucial role in RL. Exploration refers to the agent trying different actions to discover potentially better rewards. On the other hand, exploitation refers to the agent's tendency to exploit actions already known to be rewarding. Striking the right balance between exploration and exploitation is critical. One commonly used exploration strategy is epsilon-greedy, where the agent selects the action with the highest estimated Q-value most of the time (exploitation) but occasionally explores by selecting a random action with a small probability (exploration). This ensures the agent explores new actions while exploiting the ones with the highest estimated value.

2.5.4 Deep Reinforcement Learning

Deep RL combines RL algorithms with deep neural networks to learn policies that represent MDPs and solve complex problems. An important advantage of employing neural networks to represent RL policies is their ability to handle high-dimensional state spaces and generalise to similar, unseen ones. RL algorithms, such as SARSA (Sutton and Barto, 2018, p. 129) and Q-learning (Watkins and Dayan, 1992) described in the previous section, face challenges when dealing with large state spaces due to the curse of dimensionality. Deep RL provides a solution to this problem.

Deep Q Network (DQN) (Mnih et al., 2013) is a pioneering work in deep RL, as it introduced an approach for integrating deep neural networks with Q-learning. In addition to utilising deep neural networks to extract low-level features automatically, the authors also introduced the concept of experience replay. This technique effectively mitigated the impact of learning from correlated data and non-stationary distributions, resulting in increased stability. The integration of deep neural networks, experience replay, and Q-learning in DQN enabled the original authors to learn control policies for ATARI 2600 games directly from raw pixels, achieving superhuman performance.

2.5.4.1 Advanced Policy Optimisation

Trust Region Policy Optimisation (TRPO) (Schulman et al., 2015) and Proximal Policy Optimisation (PPO) (Schulman et al., 2017) are advanced implementations of on-policy deep RL algorithms. They offer an improvement over the vanilla implementation of policy gradients (Sutton et al., 1999) and differ primarily by constraining changes to the policy for a more controlled update.

TRPO aims to optimise policies by constraining the policy updates within a trust region. It iteratively maximises the expected return while staying within a specified maximum Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) between the old and new policies. This ensures that the updated policy does not deviate significantly from the old policy. TRPO optimises a surrogate objective function that approximates the expected return while considering the trust region constraint. However, although it guarantees monotonic policy improvement, it can be computationally expensive due to the need to solve a constrained optimisation problem.

On the other hand, PPO addresses the computational complexity of TRPO using a more straightforward and efficient approach. PPO performs multiple epochs of minibatch updates, where the policy is optimised to maximise a clipped objective function. The clipping mechanism constrains policy updates within a range, ensuring stability and preventing policy divergence. PPO balances sample efficiency and ease of implementation, making it popular in practice.

2.5.5 Actor-Critic

Actor-Critic methods are a class of RL algorithms that combine elements of both value and policy-based approaches. In these methods, an agent learns by simultaneously maintaining two components: an actor and a critic. The actor is responsible for selecting actions based on the current policy and directly interacts with the environment. It aims to learn a policy that maximises the expected cumulative rewards over time. The critic is the value-based component and learns to estimate the expected cumulative rewards from a given state under the current policy. The critic provides feedback to the actor by evaluating the quality of actions and states. By estimating the value function, the critic acts as a baseline for assessing the performance of the actor's chosen actions and helps to guide the learning process. Advantage Actor Critic (A2C) (Mnih et al., 2016) is an early actor-critic algorithm where the actor updates the policy based on the advantage function, representing the benefit of taking the action compared to the average action value.

2.5.5.1 Advanced actor-critic algorithms

Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), its improvement Twin Delayed Deep Deterministic Policy Gradients (TD3) (Fujimoto et al., 2018) and Soft Actor Critic (SAC) (Haarnoja et al., 2018) are advanced actor-critic algorithms. They build upon the A2C (Mnih et al., 2016) method, introducing improvements in stability, sample efficiency, and handling continuous action spaces.

DDPG combines the deterministic policy gradient algorithm with the idea of actor-critic learning. DDPG employs an actor network to select continuous actions deterministically and a critic network to estimate the value function. The key enhancements of DDPG over A2C include experience replay and target networks. Introducing an experience replay buffer for storing past experiences breaks the temporal correlations between consecutive samples, reducing the policy update variance and improving sample efficiency. DDPG also introduces target networks for both the actor and critic that are slowly updated with a soft update mechanism. This stabilises the learning process by providing more consistent target values for the critic and reducing the target's sensitivity to small parameter updates. TD3 (Fujimoto et al., 2018) improves upon the limitations of DDPG by introducing three key features. First, it employs twin critic networks for clipped double Q learning, effectively addressing the overestimation bias inherent to Q-Learning. Secondly, noise is intentionally added to the target actions during training through a process called target policy smoothing. This approach regularises the learning process and reduces sensitivity to minor policy updates. Finally, instead of updating the policy after every step, TD3 utilises delayed policy updates, resulting in less frequent network changes. This strategy helps mitigate overfitting and enhances overall stability.

Like DDPG and TD3, SAC is an RL algorithm designed for continuous action spaces. SAC balances maximising expected return and exploration by leveraging off-policy learning, maximum entropy RL, and entropy regularisation. It achieves this by estimating the entropy of the policy and adding an entropy regularisation term to the objective function. Therefore, optimising a surrogate objective function maximises both the expected return and search space exploration, yielding effective policies that exhibit diverse behaviour.

3 Literature Review

Academia (Gao et al., 2023; Lin et al., 2021; Lopera et al., 2021; Markov et al., 2015) and industry (Agnesina et al., 2023; Ho et al., 2023a; Khailany et al., 2020) heavily research tools for IC design because the production of larger semiconductors partly relies on a pipeline of automated software tools. Contrastingly PCB physical design is still a predominantly manual task, and placement-related automation is primarily limited to metaheuristics (Ismail et al., 2012; Lee, 2003; Ning et al., 2020) that study niche applications. Attempts to provide general placement solutions are not well studied in academic literature (Badriyah et al., 2017), and recent deep learning research needs robust problem formulation (Crocker, 2021) and thus has plenty of room for improvement.

In this section, we will survey PCB literature and establish state-of-the-art placement methods. Next, we will analyse the techniques used in IC physical design, limited to floorplanning, global and detailed placement and understand why IC placement tools have advanced over the past 20 years while those related to PCB have remained largely stagnant. Finally, we will briefly explore the rapidly increasing ML applications for EDA tasks and zoom into key research that is using ML or RL to solve placement tasks.

3.1 PCB Placement

PCB physical design literature, may be classified into two categories: general solutions that attempt to optimise the placement of components leading to overlap-free layouts (Badriyah et al., 2017; Merrill, 2021) with minimised wirelength and particular solutions that co-optimise particular problem parameters together with placement (Alexandridis et al., 2017; Ismail et al., 2012; Ning et al., 2020). Black box optimisation methods are widely applied in both categories because they offer the flexibility of defining complex multi-objective functions, and problem sizes are sufficiently small. In this context, we will investigate how parameters of interest are optimised and how conflicting ones are handled. We will also study different ways of formulating the problem that can simplify the problem by design.

3.1.1 General Placement

Badriyah et al. (2017) implemented a complete solution for automated PCB layout, using a Genetic Algorithm (GA) for component placement and Lee (1961) routing algorithm for making wire connections between pins. For placement, each component's spatial coordinates and orientation were directly encoded in the chromosome as a vector of triplets. The fitness function was comprised of a weighted sum across three terms: overlap, tidiness and distance between high-power components. To create a new offspring, first, the crossover operator is applied which swaps two components from the parents. The mutation operator is subsequently invoked and randomly applies a perturbation to the chromosome. This results in an alteration of the components' spatial coordinates and orientation. For simple circuits, the authors demonstrated that their placement method could provide a solution without overlap in an acceptable amount of time. The work is unique due to its application to a PCB and the treatment of both placement and routing for general PCB design tasks. However, it falls short in terms of contextual analysis and evaluation since the authors only present wirelength values without any insight connecting to the result. The latter is important since, unlike IC design, PCBs are not always constrained by layout area and minimising wirelength in such cases may not be the primary goal.

3.1.2 Thermal-Aware PCB Placement

Extensive research has been conducted to optimise the components' location on a PCB for optimal thermal operating conditions. These endeavours are driven by circuit performance and reliability in mission-critical applications such as automotive, aerospace and, military and defence. Operating at a lower temperature reduces the risk of premature component failure. Furthermore, transistors operating at a higher temperature are subject to longer signal propagation delays that may hinder operation. The problem has been studied in convectively cooled PCBs Alexandridis et al. (2017); Ismail et al. (2012); Ning et al. (2013, 2020), MCMs (Cheng et al., 2007; Lee, 2003).

Ismail et al. (2012) formulate the PCB component placement task as a multi-objective optimisation problem that considers spatial placement, device junction temperature and power-related parameters. A GA is used to optimise the fitness function comprised of a linearly weighted sum of objectives. They argue that manually tuning the fitness function is an iterative and time-consuming process, and even experts may struggle to find the optimal combination. In response, a second GA was employed to overcome this limitation. The inner loop generates candidate solutions and evaluates them with a weighted fitness function whose weights are provided by the second GA wrapped around this process. For

a circuit with 20 components and a population of 100 and 50 chromosomes for the inner and outer loops respectively, they demonstrate an 11.2% improvement over a manually tuned fitness function. The associated cost of the second GA results in triple the running time, albeit reduced to double if the population of candidate solutions is halved and a 3% deterioration in accuracy is tolerable.

The placement solutions presented so far used GAs to optimise placement and other objectives. Alexandridis et al. (2017) generate PCB layouts optimised to have an overall lower temperature. However, for searching the solution space, they use Comprehensive Learning Particle Swarm Optimisation (CLPSO), a variant of Particle Swarm Optimisation (PSO) that is less likely to converge prematurely. This is done by decoupling the velocity update of the particle solution from the global best position. The original authors further note that CLPSO can optimise multi-modal objective functions better. The advantage of PSO over GA is in terms of ergodicity or quicker exploration of the search space, which was desirable for the authors since fitness evaluation used a Finite Element Analysis (FEA) to solve for the steady-state temperature of the PCB in three dimensions. A GA discards the less fit portion of the population every generation and, from a computation perspective, is more expensive than PSO.

The original authors evaluate CLPSO relative to PSO and a GA. The original authors evaluated CLPSO relative to PSO and a GA, and over 20 runs, PSO variants exhibited lower variance when minimising the overall component temperature. There was practically no difference between PSO and CLPSO. Minimising the average temperature resulted in individual components operating outside the specified thermal region. As a result, the objective function was reformulated in a second test to optimise the temperature relative to the maximum operating temperature. PSO variants were shown to optimise the placement and distribute the temperature more evenly for the device's optimal operating point. The work highlights that PSO variants were more suitable for the task due to exhibiting smaller standard deviations while attaining statistically significant fitness improvements. They attribute the superiority of PSO variants over GAs to their ability to optimise continuous problems. Furthermore, CLPSO was shown to handle multi-modal objectives better than PSO.

The power of meta-heuristics lies in the flexibility of defining the objective function. However, analytic techniques provide an alternative optimisation method and can be used when the objective function is differentiable. In such scenarios, the objective function can be optimised using standard techniques, often with a reduced running time. Lee (2003) uses a force-directed algorithm based on the heat conduction analogy to perform thermal placement. The process starts by initialising the circuit components onto an unbounded substrate. Iteratively, each chip pushes its neighbours with force based on the heat con-

duction analogy until the solution reaches thermal equilibrium, at which time convergence would be attained.

3.1.3 PCB Placement for Power Modules

The complex electrical phenomena that arise from the sub-optimal layout in high-power applications, such as power delivery systems in electric vehicles, directly influence efficiency. From a PCB layout perspective, sub-optimal placement of high-power components may deteriorate the power supply's efficiency in the form of switching loss and affect the dynamic behaviour, a study by Ning et al. (2013) suggested. The pressure to minimise design size and increase integration may, in turn, increase coupling between components resulting from the component being placed too close to critical circuit elements or current paths. This is exacerbated when board density increases due to having multiple modules in parallel. In more technical terms, for a single converter, the parasitic effects will increase the voltage overshoot leading to power loss. For a parallel converter, the parasitic parameters must be balanced to ensure even power distribution between several sources and a single load, thus reducing localised hot spots on the PCB a later study by Ning et al. (2020) noted. As a result, the layout problem is formulated as a multi-objective optimisation problem that optimises the layout to minimise the parasitic effects and yield a legalised (Design Rule Violation (DRV) free) design. The model having the lowest power loss and no overlap between components will be the ideal candidate.

According to Ning et al. (2013), a small number of handcrafted layouts (usually fewer than 30) are created and evaluated in practice. Due to this limited sample size, it is difficult to distinguish between the designs using any quality measure. Consequently, the main drawbacks are limited options, human inconsistency, and time constraints. To overcome these limitations, two GAs in cascade are used to automate the entire design flow comprised of placement and routing. The outer GA is for placement, and the inner GA is for routing. After both loops have been completed and a legalised layout has been obtained, the design is subject to evaluation. A proxy function that considers parasitics was adopted for evaluation over FEA methods or theoretical calculations, thereby favouring computational time over accuracy. The chromosome encoding is inspired from Hatta et al. (1999), and its constituent components are relative position as a sequence pair, orientation, and gap distance. Rather than encoding the spatial coordinates, sequence pairs describe how components are relatively placed subject to vertical and horizontal constraints. The advantage that arises is that every solution is free from overlap, and the placement solution space is finite. A single-point crossover and mutation mechanisms are employed. An exception case is included when the crossover is defined within the sequence pair since it

Foundation Exploration		Modern Developments			Recent Progress		
<1990s		1990s – 2010s			>2010s		
			Analytic Techniques		Analytic Techniques		
Partitioning	Simulated Annealing	Min-Cut (Multi-level)	Quadratic / Force-directed	Nonlinear Optimization	Quadratic	Nonlinear Optimization	Advanced Techniques
Breuer *	TimberWolf	FengShui	GORDIAN	NTUPlace3	SimPL	ePlace	DREAMPlace
Dunlop and Kernighan *		Capo	FastPlace	mPL6	POLAR	RePlAce	Google *
Early Generation			Early Modern Generation		Modern Generation		Current Generation

Figure 3.1: Evolution of IC placement tools (Hao et al., 2022).

may result in an incompatible chromosome. The authors claim that results were obtained faster and with better fitness compared to manual handcrafted designs. Their final designs were evaluated using the highly accurate FEA methods. They pictorially present layout results but provide no empirical measurements other than fitness values.

3.2 Layout Techniques in IC Physical Design

Figure 3.1, proposed initially by Markov et al. (2015) and later updated by (Hao et al., 2022), illustrates the historical progression of IC floorplanning and placement tools (Asterisk denotes tools that have not been assigned a handle). It begins with partition-based placers in the 1960s that leveraged techniques from graph theory and black-box optimisation techniques (Kirkpatrick et al., 1983). In the 1980s, analytic placers were proposed, but due to a lack of computational power, meta-heuristics (Sechen and Sangiovanni-Vincentelli, 1985; Wang et al., 2000) dominated; however, they resurfaced in the early 2000s and were widely adopted due to their success on large netlists. At the time of writing, non-linear analytic placers are the current state-of-the-art (Agnesina et al., 2023; Cheng et al., 2019; Lin et al., 2021) in the literature; however, ML techniques are being extensively studied in every aspect of the EDA process (Huang et al., 2021; Khailany et al., 2020) showing potential to augment or replace particular tasks. This section briefly describes early placement techniques and reviews the mechanisms attributed to the success of the analytical placement.

3.2.1 Black Box Optimisation

Meta-heuristic algorithms were popular during the 1980s to mid-1990s when ICs contained less than 100,000 cells (Markov et al., 2015). They were effective on small cir-

cuits and even adopted by industry for commercial layouts (Sechen and Sangiovanni-Vincentelli, 1986). However, these techniques did not scale to larger circuits and were ultimately phased out. Since PCB layouts are orders of magnitude smaller, meta-heuristic approaches may provide a competitive alternative for the component placement task.

3.2.1.1 Simulated Annealing

Kirkpatrick et al. (1983) introduced Simulated Annealing (SA) as a method for combinatorial optimisation. Their work thoroughly details the algorithm and discusses problem formulations for IC physical layout problems relating to partitioning, component placement and routing. This work laid the foundations for Timberwolf (Sechen and Sangiovanni-Vincentelli, 1985) and Dragon2000 (Wang et al., 2000), both built around SA. The Timberwolf suite of tools has been used in the industry. It consists of four software programs for IC layout and routing. The two placement tools of interest for our work are the standard cell placement tool and the macro placement tool. The routing program and the placement optimiser for gate arrays will be omitted from this discussion.

The standard cell placement tool is a mixed-size placer capable of optimising the cell placement onto rows and columns, macros of varying sizes and input-output pads. The cost function comprises the linear combination of approximate wirelength and overlap sum computed from the circuit netlist, and the goal is to minimise its value. A random placement provides the initial state from which a new state may be iteratively generated. A typical cycle involves enumerating all cells, macros and pads and then drawing two random numbers that will dictate the type of operation to be performed. For instance, if both numbers represent an element of the same type (e.g. macro), the two are swapped. Otherwise, an element-wise operation limited to the element selected by the first random number is performed. If it corresponds to macros or pads, an orientation perturbation is applied, and if it is a cell, it is randomly moved to a new location. The perturbations are limited to the range in which they can be applied, and the region gradually decreases as the temperature cools.

Macro/Custom placement optimisation optimises macro and custom cell placement. The differences between the two lie in the former having known dimensions and aspect ratio, whereas the latter has bounds placed on the aspect ratio that may be altered, giving the tool additional freedom to adapt the area according to the needs of the problem. Furthermore, the designer can fix the pins of custom cells or move around the boundary as one of the perturbation steps of an annealing algorithm. In contrast with the previous program, whitespace must be allocated to accommodate wiring; otherwise, the router will be required to alter the placement or fail to find a solution. Interestingly, Sechen and

Sangiovanni-Vincentelli (1985) evaluated this tool on a PCB placement task, reporting a CPU time of 18 hours for placing a moderately complex 613-component circuit with 900 nets and 4000 pins. Subsequent routing tools managed to route 96% of the layout. Compared to manual placement, they reported a duration four-month and 99% routing.

Dragon 2000 (Wang et al., 2000), a competitor of Timberwolf, takes a top-down approach toward placement. Identifying that SA is responsible for most of the running time, they split the task into global and detailed placement. In global placement, a partitioning-based placer recursively bisects the circuit into smaller subsections. During detailed placement, the small sub-circuits are legalised and solved iteratively using a greedy heuristic. Their results on ISPD98 (Alpert, 1998) and MCNC circuits (Yang, 1991) show an average of 1.4% wirelength improvement over Timberwolf while requiring half the running time.

3.2.1.2 Genetic

Cohoon and Paris (1987) proposed Genie to solve the placement problem using a genetic algorithm. They propose multiple direct encoding schemes for the chromosome. They use a cut-paste-patch crossover technique comprised of passing and target parents. The target parent chromosome is updated from the passing parent and then patched such that the offspring yields a legal solution. Two mutation operations were defined and operated by interchanging a pair of modules. The difference lies in the selection of the modules. In one case, any two modules can be swapped, whilst in the other, only modules used within a specific net may be used for interchange. The latter tends to have a less destructive effect on the global solution because of its localised effect. Their work was compared against that of Kirkpatrick et al. (1983) on small layouts comprised of at most 50 macros and showed a 2.5% average improvement in solution quality. However, they acknowledged that Timberwolf (Sechen and Sangiovanni-Vincentelli, 1986) offered comparable or better quality while requiring an order of magnitude less CPU time. This is exacerbated by the inconsistency observed on larger layouts, where the original authors noted a 13% to 50% variation in solution quality relative to their baseline.

3.2.2 Analytic Placers

Analytical placement addresses the limitation of metaheuristics to scale to problem sizes exceeding 100,000 elements (Markov et al., 2015). They stand out by defining a differentiable objective function that at the very least has to capture wirelength and overlap in a differentiable manner. The widely adopted Half Perimeter Wire Length (HPWL) (Spindler and Johannes, 2007) is not differentiable and thus is replaced by approximations (Refer

to Appendix D.3 for a comprehensive overview of net models). Various ways for minimising overlap are suggested. For instance, the earlier placers (Brenner and Struzyna, 2005; Kleinhans et al., 1991; Viswanathan and Chu, 2005) used a spreading technique to spread out the cells and reduce overlap and congestion. However, this did not allow co-optimisation along with wirelength and required alternating passes between wirelength minimisation and legalisation. Thus, subsequent methods modelled cell placement as a spring or electrostatics system where both wirelength and overlap could be simultaneously optimised (Kim and Markov, 2012; Lu et al., 2015a; Spindler et al., 2008). Analytic placers achieve state-of-the-art performance in academic literature (Cheng et al., 2019; Guo and Lin, 2022; Lin et al., 2021) and are also utilised in the industry because they achieve the best placement quality for large-scale circuits.

The success of analytical placement in addressing scaling introduces new challenges. Highly optimised placements can suffer from congestion and fail in subsequent routing stages. Co-designing placement tools to address foreseeable problems are discussed in this section after introducing quadratic and non-linear analytic placement techniques. We will also discuss hierarchical approaches to address the intense computation demanded by non-linear optimisation methods.

3.2.2.1 Quadratic Placers

Quadratic placement alternates between unconstrained wirelength minimisation and legalisation or spreading out the cells and reduce overlap. The wirelength minimisation step optimises the layout for wirelength ignoring cell overlap. Overlap is reduced during the legalisation step. Some models ignore wirelength degradation, while others make wirelength / routability aware changes. Reducing density is done via combinational and numerical techniques including Network flows (Brenner and Struzyna, 2005), estimation of density gradients (Viswanathan and Chu, 2005) and full spreading (Kim et al., 2012a)

Partitioning quadratic placers take a divide-and-conquer approach, where both the netlist and the layout area are recursively bisected. Constituent circuits are assigned to a region. Gordian (Kleinhans et al., 1991) and BonnPlace (Brenner and Struzyna, 2005) are examples of partitioning-based placers. These methods differ from a min-cut placement (Agnihotri et al., 2003; Roy et al., 2005) in that the quadratic cost function is minimised in each step. Global performance tends to suffer due to partitioning and is further problematic for mixed-size placement since they ignore module dimensions.

Force-directed placement is a particular quadratic placement where the netlist is modelled as a system of springs. It involves three forces in the force-equilibrium equation. In the spirit of Newton's third law of motion, a spring force is used as the quadratic ap-

proximation of wire length accompanied with a corresponding opposing hold force. A density-based spreading force is also incorporated for an evening out cell density by moving cells from high-density to low-density regions. Modern quadratic placers (Hu and Marek-Sadowska, 2005) (Viswanathan and Chu, 2005) (Spindler et al., 2008) SimPL and its derivatives (Kim et al., 2012a), MAPLE (Kim et al., 2012b), and ComPLx (Kim and Markov, 2012) use the same concepts but differ in their modelling of the density force.

3.2.2.2 Non-linear Placers

Non-linear placers use non-linear optimisation techniques to find placement solutions in non-convex search spaces. Wirelength and density are modelled using smooth approximations, thus, their gradients can be computed analytically. The dominant wirelength models include the Logarithm-Sum-Exponential (LSE) model (Naylor et al., 1998), and the weighted-average model (Hsu et al., 2011), which provide a better approximation compared to the quadratic estimate. Density models include the bell-shaped function (Naylor et al., 1998), Helmholtz equation (Chan et al., 2005) and Gaussian equation (Chen et al., 2008) . Density constraints imposed on the placement grid are integrated into the objective function via Lagrange relaxation and solved using conjugate gradient or Nesterov's method. The combined optimisation function is non-convex, contrasting with quadratic and force-directed placement. The major drawback of non-linear placers is that non-linear numerical optimisation requires long run times. APlace (Kahng and Wang, 2006) and NTUplace3 (Chen et al., 2008) both used the bell-shaped density model, while mPL6 (Chan et al., 2007) used a Helmholtz smoothed density model. Multi-level cell clustering segments the netlist and is often used to mitigate the long runtimes inherent to non-linear placement techniques. However, this comes at the cost of quality degradation, which Lu et al. (2015a) suggest is not negligible while proposing ePlace.

ePlace (Lu et al., 2015a) uses a weighted average wirelength model and models density as an electrostatic system. Every object in the netlist is modelled as an electric charge leading to the density cost being the system's potential energy and its gradient, the electric field. ePlace avoids quality loss by operating on the full netlist and does not perform clustering. This novel approach brought an improvement over all the contemporary placers reporting reduced wirelength by as much as 7.13% and runtime by 3.05 times when compared to the top placers BonnPlace (Brenner and Struzyna, 2005), NTUplace3 (Chen et al., 2008), and MAPLE (Kim et al., 2012b). It failed to generate routable placements on some benchmarks, such as SUPERBLUE12, due to routing hotspots imposing routing demands twice the supply capacity. RePlAce (Cheng et al., 2019) mitigated this by adding a routability term to the objective function. ePlace is a standard cell placer and cannot han-

dle macros. For this reason, improved variations were created for mixed-sized placement ePlace-MS (Lu et al., 2015b) and for three-dimensional ICs ePlace-3D (Lu et al., 2016).

DREAMPlace (Lin et al., 2021) is the current state-of-the-art analytic placer and casts the analytic placement task as equivalent to neural network training. The authors note that prior to their work, parallelisation was achieved by partitioning the netlist. With such techniques, the achieved speedup was limited to 5x at the cost of quality degradation ranging between 2 and 6% (Li et al., 2017; Lin et al., 2015). They also point out the logistic problem: there are no standardised frameworks for solving placement tasks, and current open-sourced tools are not well managed, resulting in a high development overhead, ultimately limiting the systematic validation of novel algorithms. To mitigate these issues, the authors draw analogies between neural network training and solving the placement task, arguing that both processes perform non-linear optimisation. They leverage PyTorch (Paszke et al., 2019), a widely adopted deep learning toolkit, to implement wirelength and density estimation functions. They report a 40x speedup without quality degradation using GPU acceleration compared to RePlace (Cheng et al., 2019).

In the task of neural network training, every data instance in a dataset is fed to the neural network and prediction error is accumulated. The task of training is to minimise the prediction error with respect to the weights. The cell's spatial coordinates (x,y) are analogous to the weights, and every net in the netlist is analogous to a training data point. The label is set to zero since we want to minimise the wirelength. Therefore the netlist wirelength cost is obtained by doing a forward pass and accumulating the intermediate wirelength. The backward pass computes the error gradient to be minimised. Density is unrelated to the nets, and its cost corresponds to the regularisation term.

3.3 Machine Learning

ML is seeing broad adoption in EDA (Huang et al., 2021; Khailany et al., 2020; Lopera et al., 2021) because processes have enormous configurability while the individual processes are automated. This is illustrated in the comprehensive survey by Huang et al. (2021) that describes applications in HLS, physical synthesis, physical layout, fabrication, testing and verification. Khailany et al. (2020), takes an EDA perspective and highlights traditional techniques, machine learning, and deep learning approaches with an applications focus.

Three categories are identified where ML can impact the EDA flow mainly, predicting performance parameters, design space exploration, and AI-assisted workflows. This thesis aims to contribute AI-assisted workflows for PCB placement and excluding hybrid approaches (Ho et al., 2023b; Huang et al., 2019; Xu et al., 2022) this category is still in it

is infancy (Crocker, 2021; Mirhoseini et al., 2021). Automatic feature extraction directly from circuit netlists (Ren et al., 2022) also shows potential for improved generalisation (Lopera et al., 2021; Ma et al., 2020; Wu et al., 2021) and is attractive to our work. We see such supervised learning approaches applicable in state space encoding (Mirhoseini et al., 2021) or to more accurately gauge performance in the reward functions (Kirby et al., 2019; Xie et al., 2021; Zhang et al., 2020). In the following section, we present seminal works in the field of IC design focusing primarily on literature for learning general models and task automation in hybrid and end-to-end flows with RL.

3.3.1 Performance Prediction

Ustun et al. (2020) highlight the inaccuracy of High Level Synthesis (HLS), or C-to-RTL tools, to efficiently map mathematical operations to particular device resources (e.g. LUT, DSP) on FPGA. This limitation arises from optimisations by synthesis tools that alter the Register Transfer Level (RTL) code, albeit retaining functionality. They convert the mathematical description into a graph with nodes representing mathematical operations and edges indicating data dependencies. Node attributes include FPGA resources and mathematical precision, while edge attributes specify the cell responsible for performing a specific mathematical operation. Through inductive learning, they perform node prediction to identify the target cell type and link prediction to cluster operations within a cell.

The microarchitecture of circuits parametrised RTL code or derived with high-level tools, such as HLS, can be optimised for specific target applications. For example, Venkatesan et al. (2019) uses Bayesian optimisation, XGBOOST (Chen and Guestrin, 2016), and neural networks to navigate the design space for a CNN accelerator in a tractable manner. Features such as the number of layers, layer shape and size are used to predict the parameters that influence micro-architectural features, eventually maximising performance per area while minimising power.

Net2 (Xie et al., 2021) uses an altered GAT (Veličković et al., 2017) to estimate individual net lengths and identify the longest paths directly from a synthesised layout. Interestingly, the original authors use nets as nodes with attributes including driver's area and fan in and fan out sizes of the current node and its neighbours. Each node is connected with its fan-ins and fan-outs by edges representing common cells. Seven layouts from ITC99 (Davidson, 1999) are synthesised in ten different ways and placed using commercial tools, effectively yielding a dataset of 70 layouts. Training is performed on 60. Accuracy-oriented implementation, Net2a achieves an overall 15% improvement compared to prior work that uses GAT while obtaining a 94.6% accuracy on individual net length estimation and 92.2% ROC AUC in classifying longest path length.

Power estimation is a time-consuming pre-layout gate-level simulation. Zhou et al. (2019) developed PRIMAL, a fast and accurate power estimation for Application Specific Integrated Circuits (ASICs). MLP and CNN-based models were trained on test benches to predict the power consumption of specific circuit blocks. The model did not generalise to unseen layouts, but for the same design, it was capable of predicting power estimates under workload conditions that were not experienced in training. GRANNITE (Zhang et al., 2020) replaces the CNN with a GNN. They represent the circuit netlist as a graph using register states and test inputs as node features and toggle rates derived from simulation as labels. Compared against a commercial power estimator, it achieves a speed of 18.7x at the cost of 5.5% prediction error on unseen circuit netlists.

Khailany et al. (2020) note that detailed routing is the most time-consuming step of physical layout, and not easy to foresee whether a design will achieve timing closure without DRVs. RouteNet (Xie et al., 2018) uses a CNN to predict routability and DRC hotspots directly from post-placement layouts. Their results report similar accuracy to post-global routing estimates, albeit requiring significantly less time. CongestionNet (Kirby et al., 2019) takes this idea further by estimating congestion from the pre-placement circuit netlist. They use GAT to parse the gate-level synthesized netlist and infer local routing congestions from standard cell connectivity patterns and logic structure. The authors note that their approach substantially outperforms traditional estimation techniques (Jindal et al., 2010; Kudva et al., 2002) while requiring a running time in the order of seconds.

3.3.2 Learning Based Placers

Placement methodologies that embed ML or RL are discussed in this section. Those employing RL are of particular interest, including the following works that influence the methodologies proposed in this thesis. Goodfloorplan (Xu et al., 2022) partly uses GNNs for automatic feature extraction and learns policies to override stochastic decision-making of SA based placer. By contrast, Mirhoseini et al. (2021) propose a novel formulation of the placement task and describe an end-to-end methodology for generating circuit placement. The section will detail these methods, among other works relevant to this thesis.

3.3.2.1 Meta-heuristics Integrating an ML Predictor

Xu et al. (2022) proposed GoodFloorplan, taking a hybrid approach to floorplanning by using an RL agent in conjunction with SA. In SA, random perturbations are applied to the current state of a layout to obtain a new layout. Suppose the new layout is better, then the new state is accepted. On the other hand, as a design space exploration method, if the new state is worse, it may be accepted with a probability analogous to the cooling

temperature of the annealing process. The original authors replace this mechanism with an RL policy that decides whether or not the new state is accepted.

The iterative floorplanning problem is formulated as an MDP. The state space consists of a floorplan solution represented as a sequence pair (Murata et al., 1996), the action space specifies five unique perturbations, the transition probability is implicit since a perturbation ensures a different next state, and a reward function that promotes improvement. The objective function is based on the linear combination of the area consumed by the floorplan and the HPWL. The agent is proportionally rewarded if an improvement is made by moving from the current state to the new. If the objective degrades, a reward of zero is assigned. Since different layouts have different placement costs, the reward given to the agent is normalised against a baseline value equal to the cost of the initial state. This allows the agent to learn across any number of different layouts.

The authors generate three synthetic datasets of 50, 100 and 200 blocks with a respective amount of 50, 200 and 1000 nets. All blocks have integer dimensions; the smaller two are assigned three pins, while the remaining have six pins at the centre of each block. The model is evaluated on two MCNC (Yang, 1991) benchmark circuits and three from the GSRC dataset. An RL architecture is configured to accept features related to the circuit netlist, layout quality and the SA process. These include the objective cost of the current state, the minimum and the average cost to date, the objective cost of the new state, the perturbed area and overlapping area related to the new state and the step number. These features are concatenated with the processed netlist graph. Three successive GraphSAGE (Hamilton et al., 2017) layers followed by a mean pooling layer compress a variable-sized netlist into a fixed embedding for state encoding. The concatenated vector represents the problem state used as input to an actor-critic (Mnih et al., 2016) architecture. The discrete action space is comprised of five perturbation-style actions. Ablation tests showed that the GCN reduced wirelength by 3.1% on average and larger datasets suggested better generalisation with lower wirelength. Overall their methodology demonstrated a 3.7% lower wirelength compared to three academic floorplanners.

Huang et al. (2019) argue that design complexity driven by the increased usage of IP deteriorates routability as measured by the number of DRVs after detailed routing. Empirical data presented by the authors suggested that the placement of the macros directly impacted routability due to the complexity they encapsulate. To mitigate this issue, they propose a CNN to predict routability and use it to guide a SA floorplanner. The motivation for predicting routability stems from a miscorrelation between global routing congestion maps and resulting DRVs after detailed routing. Chan et al. (2017) and Chiou et al. (2016) attribute this miscorrelation to complex design rules of advanced process nodes.

The input of a CNN consists of a macro density map, a pin density map (informa-

tion about the available cells), and a connectivity map (nets connecting pins). VGG16 (Simonyan and Zisserman, 2014) is employed to predict routability via transfer learning due to their small data set. A training data set is generated using a macro placer of their own making based upon layouts provided in the ISPD2015 (Bustany et al., 2015) benchmark. Cadence Encounter is used to perform cell placement and routing. The number of DRVs reported by the design rule checker is used as labels. Their prediction model is trained over four circuits and evaluated on the unseen layout. The predictor was evaluated and integrated into their baseline SA macro placer. Results demonstrated that, on average, the placer generated 29% of layouts better than the top 1% in the dataset and 77% better than the top 10%. Additionally, an average wirelength reduction of 7.3% was achieved and a decrease of 2.85 times in DRVs. However, a limitation of this study is that the layouts considered for placement only contained six macros, significantly fewer than those found in real-world designs. Therefore, whether this approach can scale to netlists with hundreds or thousands of macros remains an open question.

3.3.2.2 Application in Direct Solutions

Mirhoseini et al. (2021) take a constructive approach toward floorplanning and train RL policies for sequentially placing all macros in a netlist onto the chip canvas. The standard cells in the netlist are first clustered using a hMETIS (Karypis et al., 1999) min-cut partitioning technique. They are sorted by size in descending order so that larger macros are placed first to decrease the risk of insufficient area as the floorplan fills up. The policy accepts a compressed representation of the circuit netlist and metadata as the state. It predicts the probability distribution of placing the current macro over the placement grid without violating hard constraints. The original authors consider overlap a hard constraint and thus eliminate it by design through an action masking process. After all the floorplan is generated, commercial tools complete the layout the agent is rewarded in the terminal state with the negative linear combination of HPWL, routing congestion and density.

In order to generalise to unseen designs, they focused on learning transferable representations of chips. In other words, they used supervised learning to predict the quality of partial layouts directly from the input netlist and metadata. An Edge-GNN processes the netlist graph to produce two embeddings: a compressed representation of the netlist and an embedding for the current macro to be placed. Metadata relating to the netlist and process technology are linearly transformed to produce the third embedding. The concatenation of these three embeddings, followed by a fully connected neural network, makes up the architecture of the reward predictor that shall be used to encode the policy. PPO (Schulman et al., 2017) is used to optimise the parametrised policy whose compressed is

scaled to the size of the layout region through a series of de-convolution layers.

A dataset of 10,000 placements was used for training the reward predictor generated equally from five Tensor Processing Unit (TPU) blocks. Each data point was derived from training a policy using a different linear combination of the reward parameters and random seed for diversity. Following successful regression performance, the prediction layer is removed, and the resulting network encodes the large state space. The policy was trained using a diverse dataset containing layouts with different macro types, sizes and counts. With an increasing number of blocks, it was shown that the training takes longer, but the zero-shot performance was consequently improved. The policy was trained over 20 blocks for evaluation and individually fine-tuned on the target set containing five TPU blocks. When compared with a vanilla implementation of SA (Kirkpatrick et al., 1983), it was shown that, on average, wirelength improves by 14.4% and congestion by 24.1%. In another test, the results were compared to RePlace Cheng et al. (2019), where their methods yielded good metrics for timing and congestion on all five designs, whereas RePlace (Cheng et al., 2019) failed on both in three out of five.

Encoding the state of the policy through an extra neural network adds an extra layer of complexity, albeit presenting a novel way to make state space encoding more tractable in problems with large state spaces. This work is innovative in this respect. However, it falls short on multiple fronts. Firstly, it is difficult to reproduce and evaluate their work since the authors used proprietary layouts and commercial tools despite the numerous widely available benchmark problems. Additionally, details relating to the rationale behind the methodology still need to be included, for instance, the action space encoding and the efficacy of their novel edge-GNN. Secondly, testing performance could be improved in several areas. For instance, no results described the regression of the state encoder. Additionally, the generalisation claims are dubious when suggesting fine-tuning layouts used for the evaluation.

3.4 Key Findings from Prior Work

Based on the literature reviewed in this section, we recognise the following key findings that encourage our research:

- The IC and PCB physical design processes have similar design flows but differ significantly in scale, layout density and optimisation goals. As a result, the techniques employed in these processes have diverged. While IC design primarily adopts analytic techniques, PCB design focuses on placement co-optimisation using black-box

optimisation methods. Moreover, ML is being applied predominantly in IC design, driving ongoing research in this field.

- Literature shows the SA is highly effective (Kirkpatrick et al., 1983) for optimising the circuit placements below 100,000 elements (Markov et al., 2015) while having enjoyed commercial success Sechen and Sangiovanni-Vincentelli (1985); Wang et al. (2000). While recent PCB literature often employs GA (Badriyah et al., 2017; Ismail et al., 2012; Ning et al., 2013), it has been shown that GAs do not scale as well as SA and are susceptible to high variability Cohoon and Paris (1987). Based on these findings, we will adopt an open implementation of SA (Holtz et al., 2020; Merrill, 2021) as our baseline.

- Mirhoseini et al. (2021) proposes an innovative approach to the floorplanning task using an end-to-end RL methodology. However, certain aspects of the methodology require further clarification and empirical testing. Additionally, the evaluation employed in their study is notably weak, as suggested by Xu et al. (2022) and demonstrated by Cheng et al. (2023). This study aims to investigate and adapt the methodologies proposed by the original authors for PCB component placement and seeks to provide empirical results in areas where the original work is lacking.

- The complexity of formulating differential cost functions is justified for placement in IC design. Smaller problems such as floorplanning and PCB design are still amenable to black-box optimisation, which offer flexibility and simplicity for proposing multi-objective functions. However, these methods often rely on stochastic processes and do not exhibit non-trivial placement techniques as they start from scratch every time. RL is a precise tool for learning heuristics (sequences of moves) in problems with large state spaces guided by complex non-differentiable cost functions. The added benefit is that by leveraging experience, placements can be more quickly optimised and achieve higher quality results that meet engineers' expectations.

- Literature has yet to propose an RL iterative placement methodology, yet as witnessed in this chapter, traditional methods almost ubiquitously take an iterative approach. From this perspective, the constructive methodology proposed by Mirhoseini et al. (2021) is an outlier. This thesis acknowledges this gap and proposes a novel RL formulation accompanied by two methodologies for solving the task.

4 Materials & Methods

This chapter introduces our approach to finding new learning-based methods for PCB component placement. It is divided into four sections, following Figure 4.1. The first section utilises a GNN for making graph-level predictions on circuit placement quality and serves as the state encoder for an RL constructive placer. This approach is inspired by recent AI research (Mirhoseini et al., 2021). Sections 4.2 and 4.3 explore an iterative approach to component placement, focusing on the last three objectives. Section 4.2 formulates the problem as an RL task, analysing discrete and continuous action spaces, reward signals, and four RL algorithms: TRPO, PPO, TD3 and SAC. Section 4.3 extends the problem to a multi-component setup to develop general policies for optimising PCB component placement.

Appendix A presents additional information relating to KiCad (Bautista et al., 2022), the chosen PCB layout CAD software, automated placement (Holtz et al., 2020) and routing (Lin et al., 2020) tools, libraries and setups for working with PCB data.

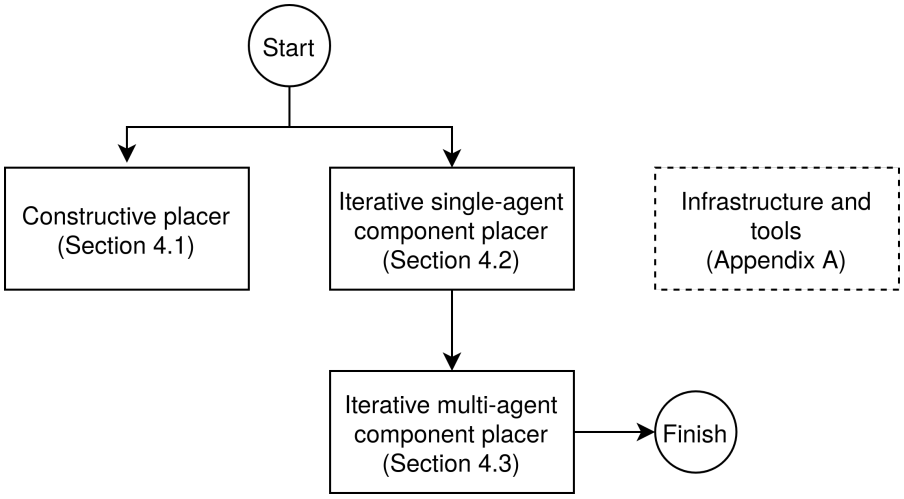


Figure 4.1: Methodology chapter flow

4.1 Constructive Placer

The constructive placement methodology was inspired by the innovative work carried out by Mirhoseini et al. (2021). The original authors reformulate the floorplanning problem as an MDP and learn a policy to place macros onto the chip canvas achieving superhuman performance. Since the PCB and floorplanning processes have some commonality, we adapt the original authors' methodology to our task.

The results quoted by Mirhoseini et al. (2021) are outstanding, but their methodology needs to be improved on numerous fronts. Basic information is missing, such as the node and edge features used in their input netlist graph. Key results are also not available to support design decisions. For instance, the proposed edge-based GNN used in the regression task of predicting circuit quality has no associated accuracy measurements. Similarly, the RL policy architecture generates a compressed embedding scaled through a series of de-convolutional neural networks for predicting the probability of placing the current component on the discrete placement grid. While their source code is publicly available, replicating their work is highly challenging (Cheng et al., 2023) because of proprietary datasets and toolchains, despite well-established alternatives (Adya and Markov, 2002; Adya et al., 2004). Based on these observations, we do not attempt to replicate their methodology directly but instead use it as a guideline. We aim to replicate the basic functionality in their method for our task while supporting critical design decisions with experimental data as we develop the circuit quality predictor and the RL policy.

In this section, we describe our purpose-built environment for constructive placement. Next, we describe our dataset derived from real-world circuits and accompanying methods for arbitrarily generating large datasets for supervised learning regression tasks. The circuit quality predictor will be presented next, which performs HPWL and post-routing wirelength graph-level predictions from circuit netlists. Finally, the wirelength predictor is embedded into the environment, encoding the policy's state space. Policies for constructively layout circuit components onto a PCB are obtained using RL.

4.1.1 Gym Environment

The OpenAI Gym (Brockman et al., 2016) environment for constructive PCB component placement is introduced in this section, with its essential elements described in Figure 4.2. The placement engine supplies the core functionality related to constructive placement. It accepts the placement probabilities generated by the policy over the discretised layout region, places the component and generates an updated circuit netlist graph. We will describe this block in detail in Section 4.1.2. The placement quality predictor derived

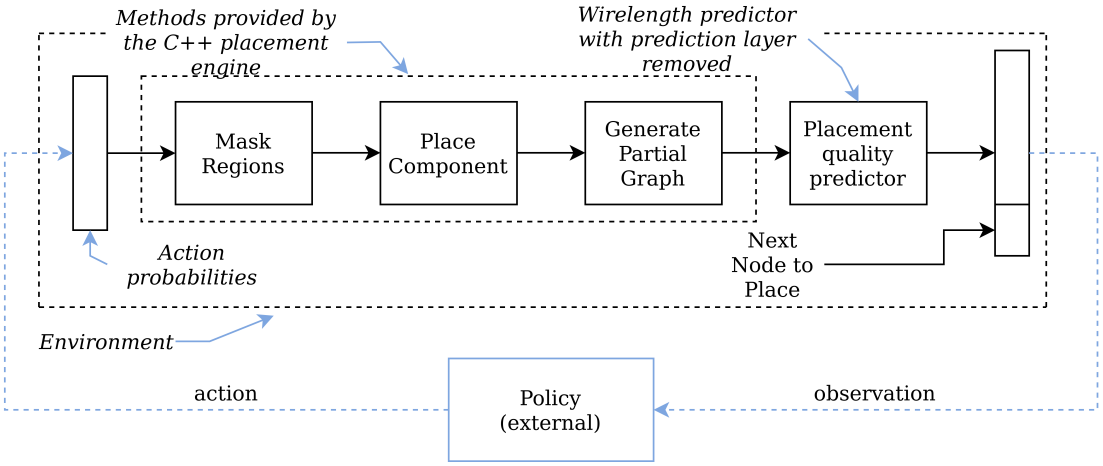


Figure 4.2: Constructive PCB placement environment. From the input placement probabilities, it masks occupied regions and places the component yielding an updated partial graph. The next observation is generated.

from a wirelength prediction model described in Section 4.1.4 maps the resulting netlist graph onto a fixed size embedding, and together with some information about the next component to place, the updated observation is returned.

The environment implements the three core methods for initialisation, resetting the environment state, and periodic stepping until the terminal condition is reached. The environment is provided with a `.pcb` file containing one or more netlist graphs and the wirelength predictor model during the one-time initialisation. During the reset phase, a netlist graph is arbitrarily selected and ordered in descending order by component area or net density. The first component is randomly placed to prevent yielding an empty observation. Ordering by area is assumed because it ensures that all the components can be placed onto the layout region and minimises the risk that, at later stages, there would be no room to place large components. Ordering by connection density prioritises the placement of components associated with the most nets. The resulting partial graph is propagated through the placement quality estimator, and together with the attributes of the following component to place, an updated observation is generated.

External to the environment, the policy maps the observation to placement probabilities over the discretised placement region, that by default, assumes a size of 20mmx20mm together with a resolution of 1mm. The placement engine processes the input action, places the component while observing the hard constraints mentioned earlier, and generates an updated partial graph representing the placement. The step method is repeatedly invoked until all elements in the ordered list are placed in the layout area. The agent receives a negative reward based on the layout's HPWL value at the terminal state and

Num	Observation	Min	Max	Type
0 - (M-1)	Compressed state based on netlist graph	0	np.inf	np.float32
M	Normalised Width	0	1	np.float32
M+1	Normalised Height	0	1	np.float32
M + 2	Pin count	1	128	np.float32

Table 4.1: Observation space for the constructive placement environment.

zero rewards in the intermediate states.

4.1.1.1 Observation Space

The observation space in the PCB component placement is based on the formulation proposed by Mirhoseini et al. (2021), albeit adapted for this specific task. It involves converting a netlist graph into a fixed-sized embedding using supervised learning. The details of this task can be found in Section 4.1.4. Together with information about the next component to place, the observation is generated having the constituents are detailed in Table 4.1, where M is the size of the fixed embedding as identified in Section 5.1.1.2.

4.1.1.2 Action Space

The action space also follows the formulation by the original authors (Mirhoseini et al., 2021). That is the policy predicts the placement probabilities over the discretised placement region. For our default discretisation described earlier the policy will predict 400 placement probabilities. The action space is bounded over the interval $[0, 1]$ and assumes a datatype of `np.float32`.

4.1.1.3 Reward Signal

The reward signal is simplified by dropping the routability and congestion terms and retaining only the HPWL approximation. Congestion is removed because our circuits are less dense in comparison to the floorplans used by Mirhoseini et al. (2021), so co-optimizing it with HPWL is not beneficial and makes the training process unnecessarily more challenging. The same line of reasoning applies to the routability term. Credit is assigned according to Equation 4.1, where r_t is the reward at timestep t for transitioning from s_{t-1} to s_t and T corresponds to the number of episode steps equal to the number of components in the netlist.

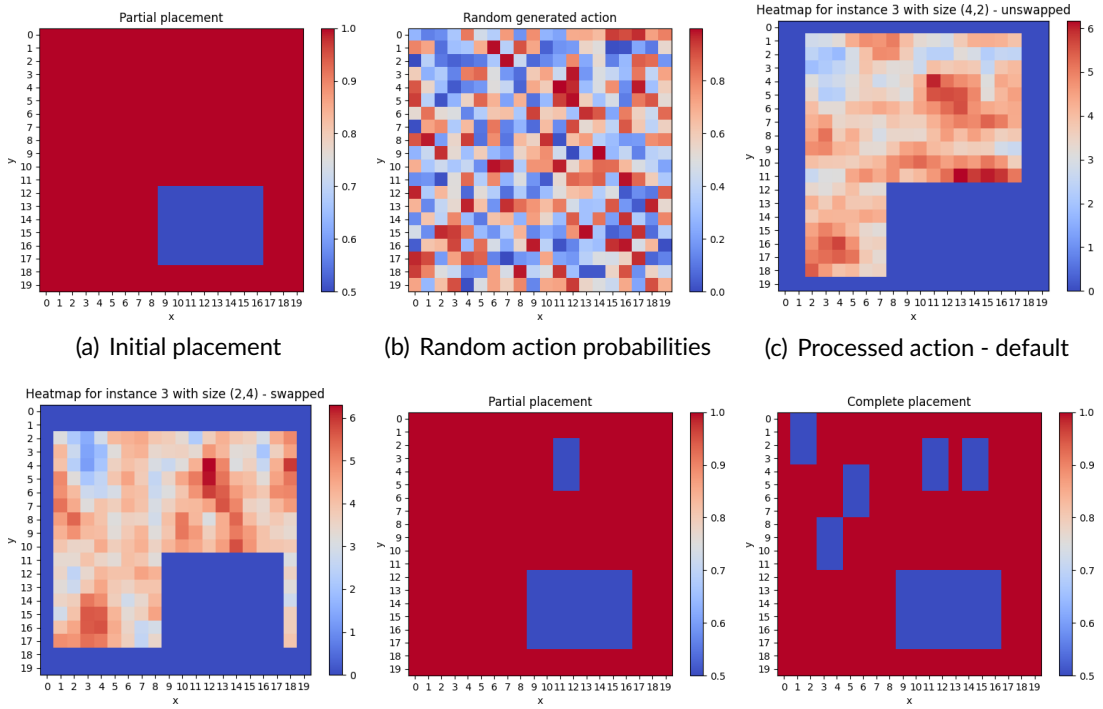
$$r_t = \begin{cases} -HPWL \; if \; s_t = S_T \\ 0 \; otherwise \end{cases} \tag{4.1}$$

4.1.2 Placement Engine

The placement engine encompasses the functionality of the constructive placer, and its basic features are described with two methods analogous to the reset and step methods of a typical OpenAI Gym environment. Miscellaneous methods facilitate the computation of performance metrics such as HPWL and interact with the operating system to save optimised circuits and log information. The placer is written in C++ and exposed to Python through Simplified Wrapper and Interface Generator (SWIG) as explained in A.5.

As mentioned, a placer object is initialised with a netlist graph representing a circuit. It operates on a discretised layout region defined by the board size that, by default, assumes a resolution of 1mm. Components rarely have integer values for their size, and to eliminate the possibility of overlap during placement, the worst case is assumed by increasing their size through rounding up. A single placement run starts by resetting the placer's state and ordering the netlist by component area in descending order. Next, the first component is randomly placed on the canvas so that a non-empty observation may be returned. The placer returns a subgraph corresponding to the placed circuit. The result of the reset state is illustrated in Figure 4.3(a). In this context, a component is placed by assigning values for position and orientation, followed by setting the locked flag indicating that the component cannot be altered further. The component's position corresponds to its centroid. It can be assigned any (x,y) location on the discretised grid such that it does not exceed the layout region or overlap with already placed components. These are hard constraints enforced by design. The orientation can be assigned an angle in increments of $\frac{\pi}{2}^c$ radians and not exceeding $2\pi^c$ radians or an equivalent value in degrees.

Outside of initialisation, the placer accepts an action generated by the policy, performs some processing to extract the best location, updates the circuit state by placing the next component and returns an observation corresponding to the placed subgraph. The action comprises a two-dimensional array corresponding to the layout region and contains the probability of placing a component at the specific discretised regions. Figure 4.3(b) is a heatmap depicting a randomly generated action. The action grid is first masked to zero out areas that would result in overlap or the component being placed outside the layout region. Next, the component is slid over the legal parts of the masked action, summing up the probabilities encapsulated by its area. The result is a new grid such as those presented in Figures 4.3(c) and 4.3(d). Notice that the scale no longer ranges between zero



(a) Initial placement (b) Random action probabilities (c) Processed action - default

(d) Processed action - swapped (e) Updated placement (f) Final placement

Figure 4.3: Key depictions of the constructive placement process. Clockwise, the graph description representing netlist is illustrated followed by its mapping onto action probabilities. Next are the processed actions followed by the updated and final placements.

and one but increases, and the component's area limits its upper bound. Furthermore, it can be observed that in both images, the value close to the layout's border is zero, as well as the region where there is the locked component from Figure 4.3(a). In actuality, one or two grids may be generated depending on the aspect ratio of the component. If the component has a non-uniform aspect ratio, two passes are made, considering the default aspect ratio and its swapped alternative. In the case of Figures 4.3(c) and 4.3(d), the component is a resistor of size 2x4mm. The grid positions with the highest value are selected for the component's position. Either way, four cases are considered, with a position corresponding to the highest value in their respective processed action grid, each associated by an orientation that differs in 90-degree increments. The one that introduces the minimal HPWL is selected for placement. Figure 4.3(e) demonstrates the resistor placement following this procedure. This process is repeated until the circuit has been completely placed. Figure 4.3(f) demonstrates a randomly placed circuit comprising six elements.

4.1.3 Dataset

Numerous published works (Badriyah et al., 2017; Cheng et al., 2022; Murphy, 2020) propose a custom PCB dataset that is not made publicly available following their research. To the best of our knowledge, no publicly available dataset suitable for PCB component placement exists. Therefore, one was constructed using 30 real-world circuits. Emphasis was placed on including various circuits, encompassing analogue, mixed-signal, and digital designs, as each type exhibits distinct connectivity patterns. Additionally, the variety was enhanced by considering the number of components, as well as their geometrical shape and type. The details of this dataset are presented in Table B.1.

4.1.4 Wirelength Prediction

The automatic generation of large circuit netlist datasets from the base set of circuits introduced in Section 4.1.3 is presented in this section. These datasets will include diverse circuit topologies with optimised placements. They will be utilised for supervised learning tasks, specifically for graph-level predictions of HPWL and post-routing wirelength. The goal of developing this model, which predicts the layout quality is to encode the problem state of a constructive placement policy as inspired from Mirhoseini et al. (2021).

4.1.4.1 Circuit Netlist

Since the netlist describes connections between component pins and we are considering a component-level graph, some pre-processing is required. This involves pruning duplicate edges that arise from different nets making separate connections across the same nodes. Furthermore, power nets are also removed since these manifest themselves as fully connected subgraphs that may impact the generalisation performance of the prediction task. Such an assumption may be justified because often PCBs have entire planes dedicated to power. As a result, a power net will manifest itself as a direct connection between the component's pin to the power plane through a VIA, effectively contributing a negligible amount to the overall wirelength. A set of attributes accompanies each node, including the component's size, coordinate position, orientation and number of pins. No edge attributes are used. The whole graph is associated with two target parameters, the approximate HPWL and the routed wirelength. Table 4.2 lists all attributes and targets and is accompanied by their data type and a detailed description. Appendix A.3 describes the object representing circuit netlists.

Feature	Feature Type	Datatype	Description
size (x)	Node attribute	Double	Component horizontal size (mm)
size (y)	Node attribute	Double	Component vertical size (mm)
position (x)	Node attribute	Double	Component horizontal position on PCB (mm)
position (y)	Node attribute	Double	Component vertical position on PCB (mm)
orientation	Node attribute	Double	Component orientation (degrees)
pin count	Node attribute	Integer	Number of component pins
HPWL	Target	Double	Derived after optimising with SA-PCB
Routed wirelength	Target	Double	Derived after routing with PcbRouter.

Table 4.2: Description of the PCB netlist graph attributes, data type and associated target parameters.

4.1.4.2 Dataset Generation

To generate the datasets, we randomly sample two sets of seed circuits from the dataset proposed in Section 4.1.3 for training and unseen testing, respectively. The training dataset will undergo a train/test split and will be exclusively used for training the model. The unseen test set will be solely used for evaluation, guaranteeing that the circuit topologies employed are not variations but completely distinct. Generating the datasets follows the procedure outlined by Mirhoseini et al. (2021), although we do not possess pre-trained policies for optimising the individual circuit and instead utilise SA-PCB (Holtz et al., 2020).

Since the task of constructive placement lays out components incrementally onto the layout region, the dataset will be comprised of netlist graphs that are partially complete ranging from a minimum of two components to a maximum, equalling the number of components in the layout. The generation flow is depicted in Figure 4.4. The process starts with a set of n unique layouts (Table B.1) in `.kicad_pcb` file format, which is converted into our internal graph representation and stored in a single `.pcb` file. The generated file is provided to the placer described in section 4.1.2, which randomly generates several partial layouts saved individually in our custom format. Layouts are sampled from a categorial distribution that favours layouts proportionately to their size. This biased sampling technique was implemented to ensure that large layouts are sampled more frequently than the smaller layout and thus allow them to contribute to a more varied dataset. Furthermore, since partial layouts are generated, sampling larger layouts more regularly will reduce the number of small partial layouts in the dataset. A log file is created for each partial layout, and the result is then converted back into `.kicad_pcb` file format and optimised with SA_PCB. The number of iterations is sampled from an integer distribution bounded by the number of components in the layout multiplied by a user-defined parameter. However, an absolute lower bound of eight iterations and an upper bound of 2048 were enforced.

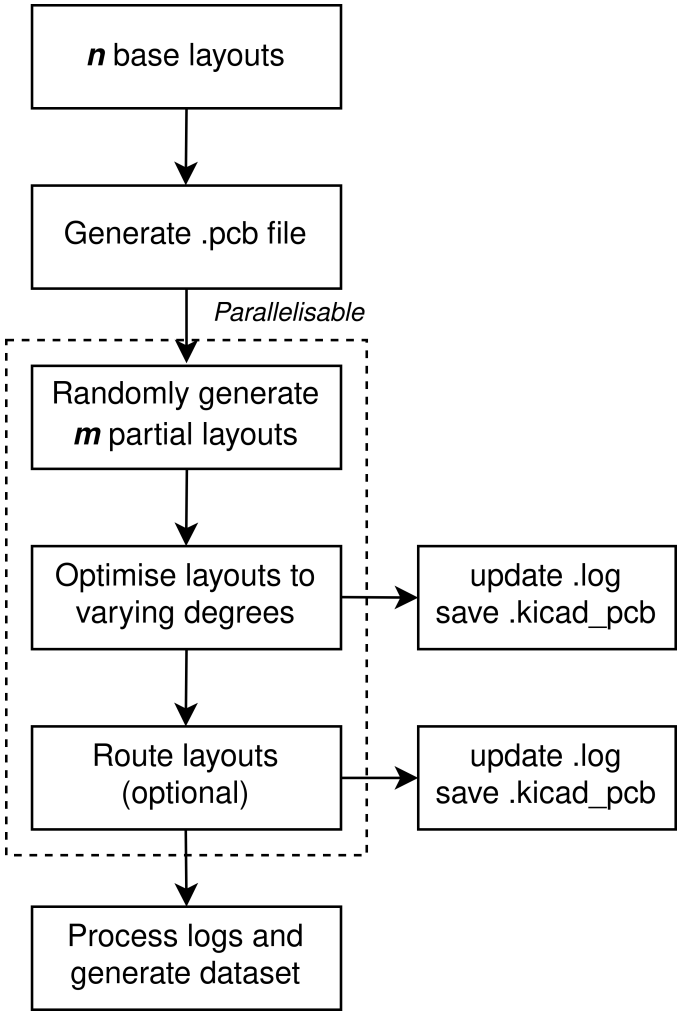


Figure 4.4: Dataset generation strategy for wirelength prediction. From a set of *n* circuits, *m* partial layouts are arbitrarily generated and optimised yielding varying lengthwise labels.

The resulting layout is saved, and the log file is updated with the placement result.

The performance metrics recorded are the approximate HPWL, overlap, and approximated RUDY congestion (Spindler and Johannes, 2007). Optionally the layouts are routed with PcbRouter (Lin et al., 2020) to obtain the actual wirelength and the number of VIAs. The routing step is made optional since the process is computationally intensive and may require a running time in the order of hours or days, and the HPWL is a good approximation of the actual wirelength (Spindler and Johannes, 2007). When employed, the router is driven hard to minimise the wirelength, with the following configurations: a high cost was associated for VIA insertion, thus forcing it to route on a single layer when possible,

and a rip-up-and-reroute strategy was employed to allow the router to remove wires in congested regions and re-attempt making the connections to find a shorter wirelength. As a final step, the log files and layouts optimised by simulated annealing are processed to generate the dataset comprising the netlist graph, target wirelength metrics, and metadata. Table B.2 describes the constituent attributes of the dataset, while Table B.3 details the configuration of the dataset generation script.

The random generation of partial layouts, placement optimisation and routing are parallelised to reduce the time to generate the dataset. The constituents are not deterministic since parallelisation is performed at the operating system level. For repeatable performance, it must be run single-threaded.

4.1.4.3 Neural Architecture

The architecture of the supervised learning model is presented in Figure 4.5. It accepts a graph with an arbitrary number of nodes and an adjacency list. Depending on the target selected during training, graph level predictions for HPWL or routed wirelength are yielded at the output. The input graph is propagated through several convolution layers, each transforming the nodes attributes as a function of their neighbours. The graph is then collapsed into a single embedding comprised of the concatenation of the mean and max pooling operations. The fixed-size embedding is then used as an input to a fully connected MLP for prediction.

The graph-level wirelength prediction task is set up as a regression task. The dataset

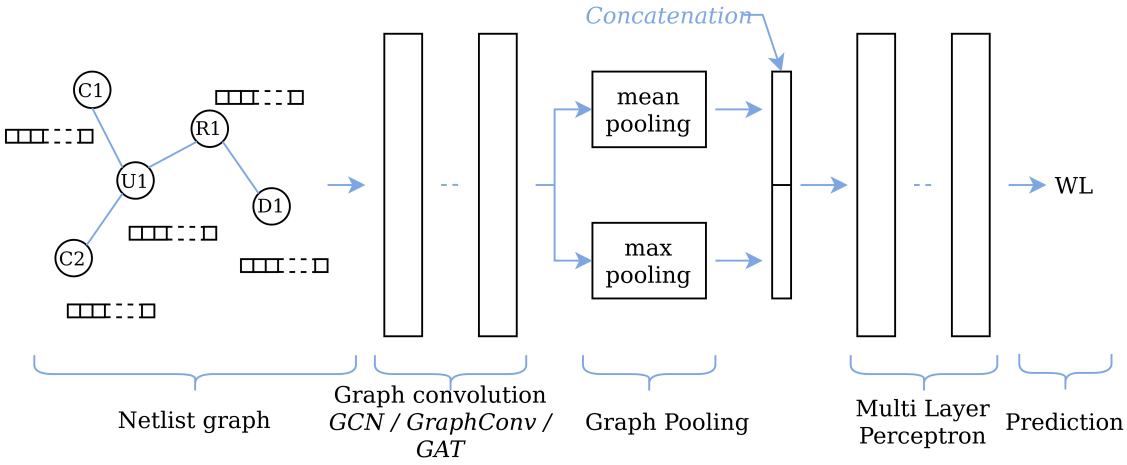


Figure 4.5: Neural architecture for wirelength prediction. The topology comprises graph layers for automatic feature extraction, pooling layers for reducing the graph to a fixed embedding, and linear layers for predictions.

of 8192 samples is split into 70%:30% training and testing, respectively and set up using Pytorch Geometric (Fey and Lenssen, 2019) data loader. The latter facilitates the handling and manipulation of variable-sized graph data. Table 4.2 describes the node attributes used as input features and wirelength targets to be predicted. Since all attributes span over an extensive range of values, they are normalised. Such operation reduces the propagation of large error gradients that may result in unstable training (Bishop, 2016).

Since the dataset is generated with a strict lower bound of four iterations, this prevented the generation of graph netlist with extreme wirelength often arising from random placement. Furthermore, since the target values were normalised, the squaring operation would significantly reduce the magnitude of the error. For these reasons, Root Mean Square Error (RMSE) loss function was selected because our dataset does not contain outliers that can singlehandedly influence the loss value. The square root operation balances the squaring process and prevents minimal loss values.

4.1.4.4 Experimental Setup

Three graph layers are tested. GCN (Bruna et al., 2013) and GraphConv (Micheli, 2009) were selected because they are architecturally different, with the former being derived from spectral graph theory and the latter motivated by information propagation. The third is GAT (Veličković et al., 2017) and was selected due to its ability to learn a method for identifying the relative importance of a node's neighbours through the attention mechanism. The optimal architecture for each graph layer is identified through 128 hyperparameter optimisation trials. The hyperparameters considered for optimisation and their bounds are noted in Table 4.3, and the experiment flow is depicted in Figure 4.6. The setup gauges the performance of each trial averaged over four consecutive runs outlined in blue with distinct albeit deterministic seed values. Optuna (Akiba et al., 2019) with a

Hyperparameter	**Type**	**Range**
Learning rate	Double	0.0001 - 0.001
Batch size (2^n)	Integer	5-8
Graph convolutional layer type	String	GCN, GraphConv, GAT
Number of graph convolutional layers	Integer	1-4
Graph convolutional layer embedding size	Integer	16-256
Graph convolutional layer activation function	String	tanh ReLU
Number of dense layers for the multi-layer perceptron	Integer	1-4
Multi-layer perceptron layer size	Integer	16-256
Multi-layer perceptron activation function	String	tanh, ReLU

Table 4.3: Optimisation criteria for wirelength prediction model.

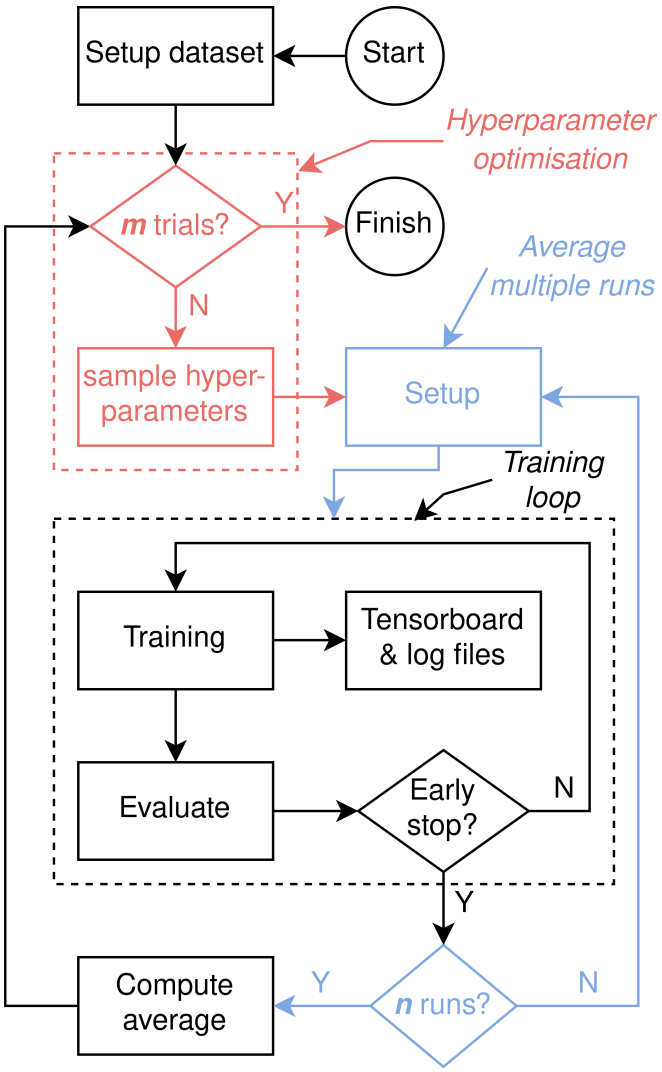


Figure 4.6: Procedure for optimising neural architecture and hyperparameters. m-trials are generated by Optuna's TPE sampler (Bergstra et al., 2011), each setup is invoked n times and averaged.

Tree-structured Parzen Estimator (TPE) sampler (Bergstra et al., 2013, 2011) is used to cleverly select hyperparameters guided by past performance.

4.1.5 Training and Experimental Setup

Due to the large and continuous action space, TRPO and PPO (Schulman et al., 2015, 2017) RL algorithms are used to learn a policy. The latter is also the training algorithm chosen by Mirhoseini et al. (2021). The policy will map the embeddings generated by the placement quality estimator and information about the next component to placement

Parameter	TRPO	PPO	Description
Learning rate	1e-3	3e-4	Learning rate for Adam optimiser
Optimiser	Adam	Adam	(Kingma and Ba, 2014) Adam
steps	2048	2048	Number of steps collected from a single rollout
batch size	128	64	Minibatch size for gradient updates
epochs	10	10	Number of epochs for optimising the surrogate loss over the current dataset
train frequency	2048	2048	Frequency in steps at which the neural network is updated $(steps * environments)$
gradient steps	160	320	Gradient steps performed per update. $(steps/batch\_size * epochs)$
environments	1	1	Number of simultaneous environments
gamma	0.99	0.99	Discount factor
Shared layers	[64]	[64]	Dense layer shared between the policy and value networks
Policy layers	[64,128,256,128,64]	[64,128,256,128,64]	Policy network architecture
Value layers	[128,256]	[128,256]	Value network architecture
Activation function	Tanh	Tanh	Activation function

Table 4.4: TRPO and PPO configurations for constructive placement.

probabilities over the discretised layout region. Stable Baselines3 (Raffin et al., 2021) is used to access performance implementations of the aforementioned on-policy algorithms. We mirror the default parameters set up by the authors of Schulman et al. (2015) and Schulman et al. (2017), respectively. Table 4.4 summarises the experimental setup.

The reward function described by Equation 4.1 returns the absolute wirelength of the circuit, and since no normalisation has been used, training is restricted to a single layout. The original work by Mirhoseini et al. (2021) performs training across numerous layouts without employing normalisation, raising concerns about their claims for generalising to unseen circuits. We randomly select three layouts from the dataset described in Section 4.1.3, use one for training and then evaluate the resulting policy on all three in terms of both HPWL and post-routing wirelength as described in the next section.

4.1.6 Evaluation

For fair evaluation, a method for capturing performance regardless of the training procedure is needed. One approach is to use HPWL approximation or actual post-routing wirelength. The latter is generally preferred Spindler and Johannes (2007) as a good HPWL value does not guarantee a 100% routable layout (Cheng et al., 2022; Lin et al., 2021). For instance, Cheng et al. (2022) evaluate the effectiveness of their placement initialisation algorithm in terms of post-routing wirelength, albeit they make use FreeRouting (Wirtz, 2023). Lin et al. (2021) also use post-routing wirelength to evaluate their routing algorithm alongside a beta version of the commercial router deepPCB (DeepPCB, 2023) and FreeRouting. The number of VIAs is also reported in this case. A SA placement methodol-

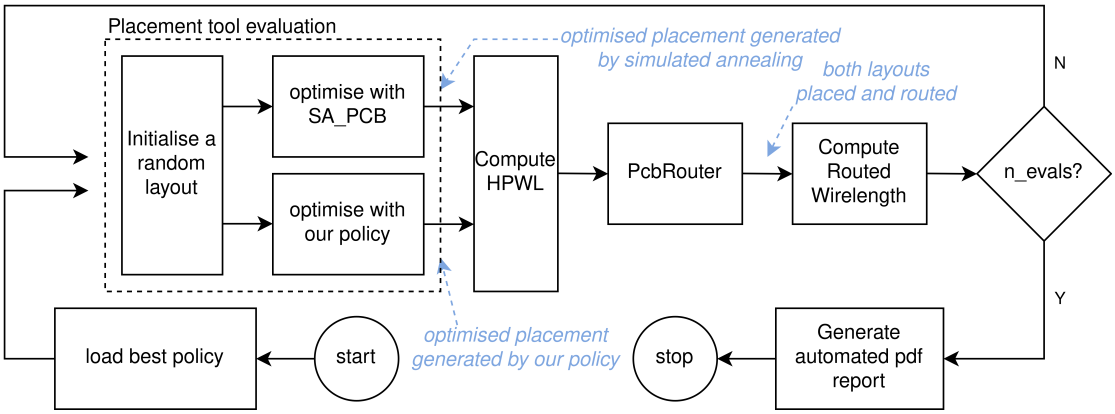


Figure 4.7: Procedure for evaluating RL policies against SA. A randomised unseen circuit is concurrently optimised by the proposed method and SA-PCB. The resulting optimised layouts are routed with PcbRouter.

ogy is used as the baseline due to its effectiveness on small circuits and immunity to variability (Kirkpatrick et al., 1983; Markov et al., 2015; Sechen and Sangiovanni-Vincentelli, 1985). SA-PCB (Holtz et al., 2020; Merrill, 2021) is employed for establishing a baseline for optimised placements, and post-routing wirelength is generated using PcbRouter (Lin et al., 2020), ensuring a fair evaluation procedure for different objective functions.

Figure 4.7 illustrates evaluating an RL policy along a baseline established by SA using post-routing wirelength. The process begins by loading the best policy for a given setup and noting the initial configuration to ensure identical starting conditions for the proposed method and SA-PCB. The HPWL of the optimised layouts are then computed, and they are subsequently routed using PcbRouter, incorporating the changes mentioned in Appendix A.2. This process is repeated four times with different initial conditions. Finally, log files generated by the evaluation runs are automatically parsed, with the mean and standard deviation tabulated in a pdf report for analysis.

4.2 Single-Component Iterative Placer

This section treats the second and third objectives by formulating the iterative PCB component placement problem as an RL task, then designing experiments for analysing the fundamental mechanisms of the problem. We aim to identify the best method to train an RL agent to orient a movable component in an otherwise locked circuit. Specifically,

1. PCB placement problem is formulated as an RL task. It provides a description of the problem setup, including the environment, observation and action spaces.

2. Four distinct reward functions are described, of which three differ fundamentally.
3. Finally, the dataset is discussed, followed by the experimental setup for training and evaluation processes, including hyperparameter optimisation and an assortment of tests aiming to empirically identify the best setup.

We focus on a subset of features typically employed in PCB design flow. We do this because we are interested in identifying the essential relationships between what works and what does not. For these reasons, we make the following assumptions:

1. Single-sided placement restricting placement to the top layer, as opposed to double-sided placement, where components are placed on both top and bottom layers.
2. Movable components must be two terminal devices (e.g. resistors, capacitors, diodes), while fixed components can have an arbitrary number of pins.

4.2.1 Iterative PCB Component Placement as an MDP

This section introduces the iterative PCB component placement concept as an MDP. We briefly outline the key elements constituting the MDP framework, including the state space, action space, state transition, and reward.

1. *State space* - A combination of the component's perceived surroundings, directional data pointing towards the goal region and neighbouring component cluster, and position and attitude information of itself on the placement region.
2. *Action space* - Discrete or continuous spaces define changes in the component's position and orientation. The latter has granular control over the magnitude and direction of movement. The former's movement is limited to discrete steps along the basis vectors.
3. *State transition* - A change in movement or orientation will cause the agent to move to another state.
4. *Reward* - The goal for generic PCB component placement correlates with an overlap-free placement having minimal wirelength. We present various formulations to directly or indirectly learn generalisable techniques achieving this.

Num	Observation	Min	Max	Type
0-7	Normalised overlap contribution in 45 segments	0	1	np.float32
8-15	Normalised line-of-sight contribution in 45 segments	0	1	np.float32
16	Magnitude of direction-of-movement vector	0	1	np.float32
17	Angle of direction-of-movement vector	0	1	np.float32
18	Magnitude of direction-of-movement vector	0	1	np.float32
19	Angle of direction-of-movement vector	0	1	np.float32
20	Normalised component centroid x-coordinate	0	1	np.float32
21	Normalised component centroid y-coordinate	0	1	np.float32
22	Component orientation	0	1	np.float32

Table 4.5: Observation space for iterative component placement.

4.2.2 Gym Environment

The environment for iterative single-component placement was explicitly designed to adhere to the specifications of OpenAI Gym (Brockman et al., 2016). Among the core methods, the initialisation function configures the environment based on user-provided settings. The reset and step methods are utilised for episodic training. Optional methods for visually rendering the environment and user interaction were omitted, and instead, OpenCV (Bradski and Kaehler, 2008) was used to export episodes as MPEG4 video files.

During the *initialisation* phase, the environment is configured with user arguments, and submodules are initialised. During the *reset* step, the `.pcb` description file containing one or more circuits is parsed, and a layout is randomly selected and initialised. If the data augmenter module (Section 4.2.6.2) is enabled, a random rotation and translation are applied to the locked portion of the circuit, and finally, the movable component is randomly initialised within the PCB area. An observation capturing the initial state is computed. Via the step method, the policy maps it onto an action which is subsequently applied to the environment, translating and rotating the movable component. Consequently, the environment returns a new observation that reflects the updated state accompanied by a reward gauging the quality of the action. The episode terminates after the agent carries out a predefined number of steps $T = 200$.

4.2.2.1 Observation Space

The observation space consists of a 23-element vector divided into three categories: the perceived surroundings that detect overlapping and neighbouring components (elements 0-15), goal information for locating the target and neighbouring components (elements 16-20), and self-positioning and orientation information (elements 21-23). Table 4.5 summarises the observation space, while each element is detailed within Section 4.2.3.

Num	Action
0	Step along the positive y-axis
1	Step along the positive x-axis
2	Step along the negative y-axis
3	Step along the negative x-axis
4	Offset orientation by 90 °
5	No change

Table 4.6: Discrete action space for iterative component placement.

Num	Observation	Min	Max	Type
0	Magnitude of translation vector	0	1	np.float32
1	Angle of translation vector	0	2π	np.float32
2	Orientation (see Equation 4.3)	0	1	np.float32

Table 4.7: Continuous action space for iterative component placement.

4.2.2.2 Action Space

Two action spaces are available for the environment, which describe the translation and orientation of the component. The discrete action space, summarised in Table 4.6, and detailed in Section 4.2.4.1, limits translation along the basis vectors and allows orientation in 90° increments. On the other hand, the continuous action space, described in Table 4.7 and detailed in Section 4.2.4.2, offers more precise control by describing translation as a vector while resolving orientation orthogonally. It further differs from its discrete counterpart by allowing simultaneous translation and orientation of the current component.

4.2.2.3 Reward Signal

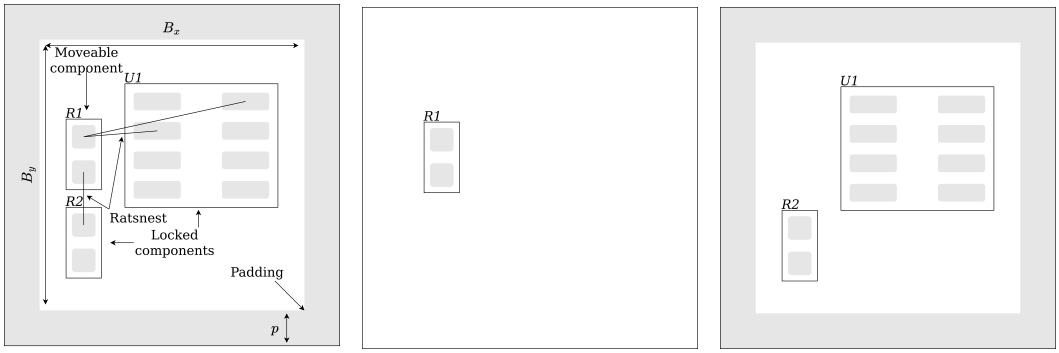
The reward signal is categorised into three classes, each presenting a paradigm shift in the behaviour promoted. Table 4.8 summarises each class and the compatible action space. Section 4.2.5 presents the mathematical formulation and discusses each class in detail.

4.2.3 Observation Space

The observation is the most complicated computation carried out by the environment. It contains three distinct pieces of information: the agent's view of its surroundings, directional information pointing towards the goal and information related to the agent's location on the PCB. The resulting observation is the 23-element vector summarised in Table 4.5. This section details the individual components making up the observation.

4.2.3.1 Surrounding View

The PCB graph representation is drawn as a grayscale image comprised of two layers to identify the surrounding obstacles. Figure 4.8 illustrates our graphical representation of a PCB along with decomposed layers. The first layer depicted in Figure 4.8(b), contains the drawing of the moveable component, while the second layer shown in Figure 4.8(c) contains the remaining locked portion of the circuit and padding. The size of both images is identical. The dimensions of the constituents in pixels are computed as a function of the board size, padding and resolution. The padding and resolution are parameters defined within the environment and are typically fixed across experiment sets. The padding is set to a value exceeding the largest dimension of any moveable component, and the resolution value is significantly smaller than the maximum allowable step size. Figure 4.8(a) illustrates the complete representation, including designators, ratsnest and dimensional information. Additional optional layers include information useful for debugging and troubleshooting such as component designators, ratsnest and component pads.



(a) Combined layered PCB (b) Moveable component (c) Fixed circuit and padding

Figure 4.8: PCB netlist representation as a stack of images. The Leftmost Figure depicts the entire layout, which is further divided into movable and locked components.

Description	Notation	Equation	Action Space
Reward based on layout dependent parameters	R_1	4.5b	Discrete
Reward based on expert knowledge, variant 1	R_{2a}	4.7a	Both
Reward based on expert knowledge, variant 2	R_{2b}	4.8a	Both
Reward based on agent's expertise	R_3	4.9a	Continuous

Table 4.8: Summary of reward signals used for iterative single-component placement.

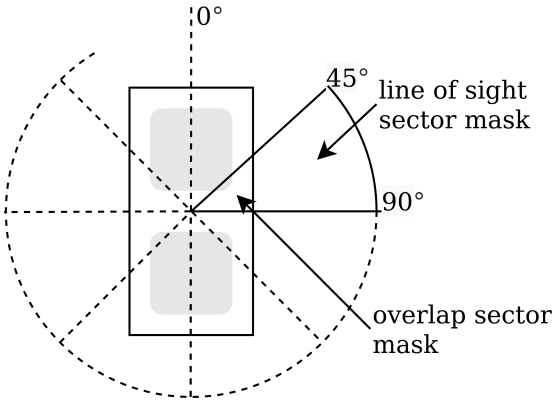


Figure 4.9: Overlap and line-of-sight masks derived from circle segments

Mask overlays are generated next to compute the line of sight and overlap data between the moveable component and the environment. A circle is drawn with a centre corresponding to the movable component's centroid and comprised of eight segments. Each segment occupies a distinct grayscale image with the same size and resolution as the pictorial in the previous section. The result is a stack of eight images, each corresponding to a 45-degree segment originating at the component's central location.

Line of sight information is derived from a series of boolean operations between the masks and the fixed layout. First, a NAND operation is performed between the moveable component and the sector to obtain the mask. This operation captures the obstacles near the component within the mask's region. An AND operation with a fixed layout follows to capture the nearby objects within the sector. The overlap measurement is similarly computed. The mask comprises the sector portion that intersects the fixed component; thus, the mask is generated by an AND operation between the circle's segments and the image containing the fixed component. The result is then applied to the image containing the fixed portion of the layout and padding. This operation captures the part of the component that intersects with its fixed environmental obstacles. Figure 4.9 illustrates how the mask set is derived from a circle originating from the movable component's origin. The circle's radius is a multiple of the component's largest dimension.

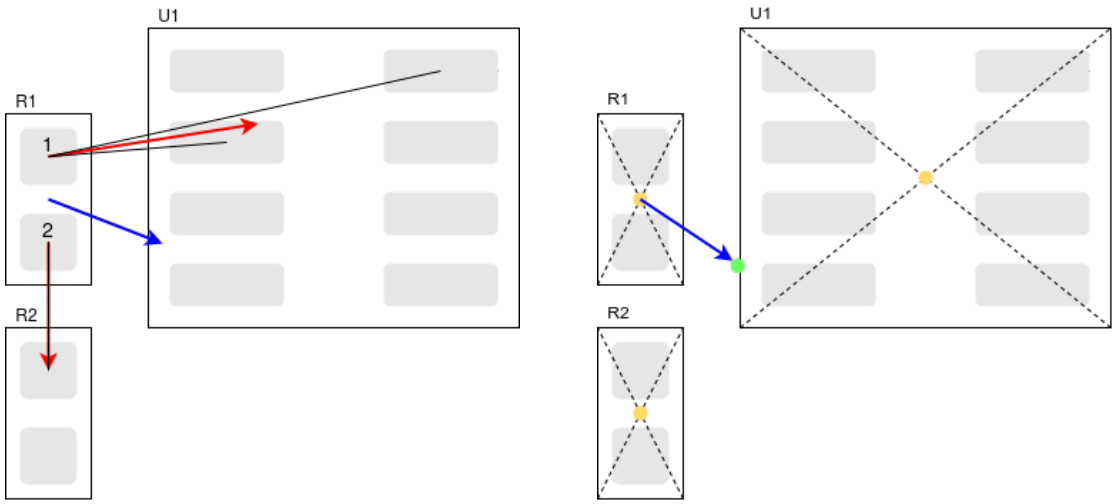
All boolean operations are performed in a bitwise manner on raw pixel data. In both cases, the resulting images of the masked regions are normalised. The images are summed up and divided by their masks to generate a normalised value ranging from 0 to 1. The drawings are generated using OpenCV (Bradski and Kaehler, 2008) in Python. The resolution of the drawings is a user-definable parameter and was set to 100μm for all experiments carried out throughout this dissertation.

4.2.3.2 Directional Data

The line of sight and overlap measurements provides the agent with information about its surrounding environment. For the agent to be able to move close to its neighbour components sharing the same nets, it needs directional information. As a result, two vectors in the form of (r, θ) are provided.

One vector is derived from all the pad-to-pad vectors between the current component and its neighbours. It roughly sums up all the vectors to obtain a mean vector that, moving along, may reduce the wirelength for all nets involved. In Figure 4.10(a), pad one of component R1 is part of a multi-pin net as indicated by the black lines of ratsnest. In this case, the resultant vector is computed, and its magnitude is divided by the number of vectors involved to maintain relatively small numerical values and have a local effect rather than a global one. The red vector represents the vector containing this compressed representation. Pad two of component R1 is a point-to-point connection; therefore, the vector is used as is, as shown by the red vector superimposed on the black connection line. The blue vector represents the feature vector and is simply the sum of the two red vectors translated to the centre of the component.

The group vector, illustrated in Figure 4.10(b) by the blue arrow, is computed between the current component's centre point and the group's centroid. It provides information



(a) Direction of movement vector

(b) Group vector

Figure 4.10: Extraction of directional information from the circuit netlist. On the left is the resultant vector obtained from all point-to-point connections between the current component and its neighbours, while on the right is the vector relative to the cluster centroid tied by common nets.

about where the agent is relative to its neighbours. Since a net is not limited to a connection between two pads (point-to-point) and may connect an arbitrary amount of pads across an arbitrary number of components, having information related to the group of all involved may benefit the agent.

$$(\bar{x}, \bar{y}) = \frac{1}{|N|} \sum_{n \in N} (x_n, y_n) = \left(\frac{1}{|N|} \sum_{n \in N} x_n, \frac{1}{|N|} \sum_{n \in N} y_n \right) \tag{4.2}$$

Equation 4.2 calculates this vector and provide the agent with information to locate the component cluster that it is part of and move in such a way that minimises the wirelength locally. N is a set of n nodes comprised of the current node and its neighbours. $(\bar{x}, \bar{y})$ is the average coordinate of all node centroids, (x_n, y_n) in the set N.

4.2.3.3 Component Information

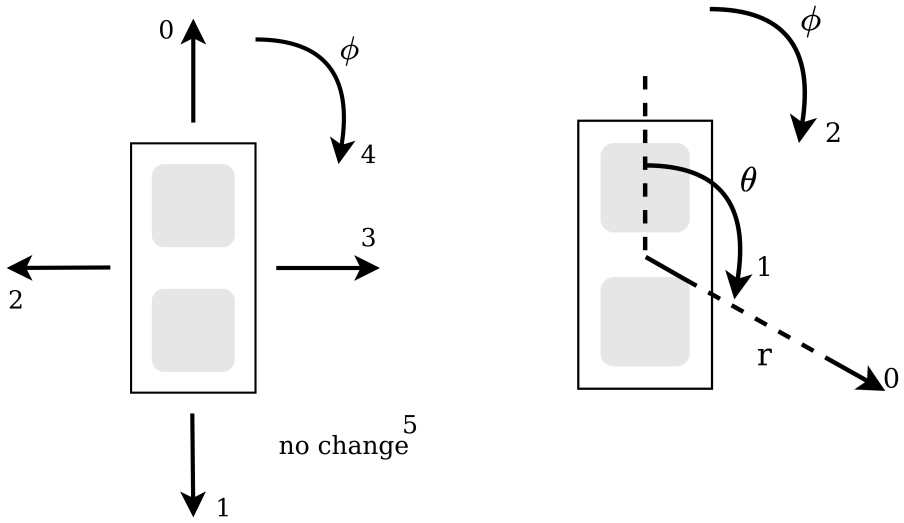
Component information is the third and final piece of the observation. This includes the node's position normalised by the board size and its orientation in radians ranging between $-\pi^c$ to π^c. Of particular interest, the position may be helpful for the agent to understand when the component is outside the board region. While the agent can partially move the component outside the board region if the whole component is outside the board region, the episode terminates early, and the agent will receive a significant penalty proportional to the remaining steps. The orientation may be helpful to make sense of the line of sight and overlap data since these are computed relative to the component orientation. If the component is orientated at 0°, the first element of the line of sight will correspond to the sector ranging from 0° to 45° degrees. Alternatively, if the orientation is 90° degrees, the first element will correspond to the 90° to 135° sector. Therefore, if its orientation changes, the vector must be interpreted accordingly.

4.2.4 Action Space

The action space defines the output of the policy and specifies the range and size of the action that can be performed. This section specifies how the agent interacts with the environment to alter its state using discrete and continuous action spaces which are respectively summarised in Tables 4.6 and 4.7.

4.2.4.1 Discrete Action Space

The discrete action space defines six actions the policy can select to perform in the environment. Concerning Figure 4.11, actions zero through three select a translation action



(a) Discrete action space (b) Continuous action space

Figure 4.11: Discrete and continuous action spaces

where the component is displaced forward, backwards, left or right, respectively. The magnitude of displacement is an environment variable that can assume values in the region of 0.3-0.7mm and, by default, takes a value of 0.5mm. These values are suggested by supplementary experiments in Appendix C.2.1. Action four offset the orientation, ϕ of the components by 90°.

4.2.4.2 Continuous Action Space

The continuous action space defines three elements the magnitude r, direction θ and orientation ϕ. The translation is defined as a vector having a magnitude and direction in the form of (r,θ). The magnitude can take values between zero and one, while the direction is defined in the range of 0^{c} to $2\pi^{c}$. Translation components are illustrated by arrows zero and one in Figure 4.11(b). The component's orientation, ϕ shown by arrow two, is also a continuous value in the range of zero to one, albeit is interpreted as one of four discrete orientations as denoted by Equation 4.3. The magnitude is scaled by the step length, an environment parameter assuming a default value of 1, effectively limiting the agent's movement to 1mm in any direction.

The continuous encoding scheme has two advantages over its discrete counterpart. Firstly, translation is not discretised; thus, the agent has more granular control. Secondly, translation and orientation can be performed simultaneously. These benefits comes at the cost of an increasingly challenging training process.

$$orientation = \begin{cases} 0^\circ \; when \; 0 \leq \phi < 0.25 \\ 90^\circ \; when \; 0.25 \leq \phi < 0.5 \\ 180^\circ \; when \; 0.5 \leq \phi < 0.75 \\ 270^\circ \; otherwise \end{cases} \tag{4.3}$$

4.2.5 Reward Signal

The reward signal is crucial for the agent to learn generalisable placement techniques that can be applied to unseen circuits. It is the medium through which the designer communicates the task's goals to the agent, effectively guiding the learning process by rewarding good actions. Four reward mechanisms are investigated, with incremental improvements in the interest of generalisation. These reward functions are classified into three fundamentally distinct groups, with the second having two variations:

1. The reward is encoded using absolute problem-dependent parameters of wirelength and overlap and enables finding solutions for individual layouts.
2. The reward encodes the expert placement as its target in a problem-independent signal. It mimics the expert designer by using expert placement as the goal.
3. The reward function is based on problem-dependent parameters of wirelength and overlap. However, wirelength is normalised relative to the initial state and best historically known values. Therefore as the policy improves, the reward signal adapts, allowing the agent to learn through self-improvement without expert guidance.

We will present the details of each reward scheme in the upcoming subsections. First, an early termination penalty common to all reward signals is introduced. Subsequent sections detail each reward category as listed above.

4.2.5.1 Early Termination Penalty

The agent can freely move the component anywhere on the XY plane representing the layout region, and the environment imposes no bounds. In other words, the component can move outside the board area, into the padding and outside the scope of the problem. While this does not cause any problems, it is a behaviour that we want to impede and would like the agent to operate inside the board region. Therefore, the agent is heavily penalised when moving the component outside the layout area. While some overlap with the padding is allowed, if the component moves entirely outside the layout area, the

episode terminates immediately, and the agent is assigned a negative reward proportional to its remaining steps. This part of the reward signal is the early termination penalty, ζ and is common throughout all reward functions. Equation 4.4a illustrates how ζ is computed, where s_m and s_t are, respectively, the maximum episode steps and the steps carried out by the agent at time t. $p_r = 16$ is a constant for scaling the step difference leading to the penalty and is assigned its values according to Equation 4.4b. Its value was identified empirically as part of preliminary testing.

$$\zeta = (s_m - s_t) \times p_r \tag{4.4a}$$

$$p_r = \begin{cases} p_r \; if \; done \; and \; s_m \neq s_t \\ 0 \; otherwise \end{cases} \tag{4.4b}$$

4.2.5.2 R_1: Reward based on Problem Dependent Parameters

Credit is assigned based on changes in wirelength, overlap and movement. Preliminary tests suggested that rewarding relative changes instead of absolute parameter values caused the agent to learn. However, in certain situations, the agent was observed finding sneaky ways to perform well without simultaneously reaching our goals. For these reasons, the R_1 described by Equation 4.5b assigns both immediate and terminal rewards.

The former is listed in Equation 4.5a and is comprised of a linear combination of HPWL h_t, overlap o_t, and movement penalty s_t. The subscript t denotes the episode timestep. The remaining parameters with an s subscript are designer-assigned constants for emphasising particular aspects of the reward signal. The terminal reward is assigned by Equation 4.5b and completes credit assignment R_1.

$$q_t = -(h_s(h_t - h_{t-1}) + o_s o_t + s_s s_t) - \zeta \tag{4.5a}$$

$$R_1 = \begin{cases} q_t - (10 \times h_t) \; if \; done \; and \; s_m \neq s_t \\ q_t \; otherwise \end{cases} \tag{4.5b}$$

Equation 4.6a gives the contribution of overlap in the reward signal. It is assigned a zero value if the sum of all overlapping sectors is less than $1e^{-3}$. Otherwise, it is raised to the power of e. Since the overlap term is stored as a double, any values smaller than $1e^{-3}$ are considered zero. This is necessary because the zero conditional tests may sometimes fail if the term contains a residual value resulting from the finite precision of the IEEE754 floating-point numeric system.

$$o_t = \begin{cases} e^{overlap} \; if \text{ overlap } \geq 1e^{-3} \\ 0 \; otherwise \end{cases} \tag{4.6a}$$

$$s_t = \begin{cases} 0 \; if \text{ no step} \\ 1 \; if \text{ translation step} \\ 4 \; if \text{ orientation step} \end{cases} \tag{4.6b}$$

The term s_t in Equation 4.5b penalises the agent from moving in the environment. Different actions acquire different penalties, as detailed in Equation 4.6. Changes in the orientation are penalised significantly higher than translation moves, and the remaining still carries no penalty. Such a mechanism can promote subtle undesired behaviours. On the one hand, it incentivises the agent to find and stick to a suitable location. On the other, it may inhibit it from moving due to being penalised. In order to ensure the former, a terminal reward is assigned proportional to the quality of the layout, thereby ensuring that if the agent decides not to move, it will perform poorly in comparison.

4.2.5.3 R_{2x}: Reward based on Expert Knowledge

The credit assignment presented next is radically different from that proposed in the previous section. It discards all task parameters for wirelength and overlap and uses the expert's cartesian position and orientation as the agent's target. The reward signal presented in this section motivates the RL agent to mimic the behaviour of the expert designer. We expect the agent to learn fundamental placement techniques given sufficiently large and diverse datasets as a side effect.

$$R_{2a} = tan\left(clip\left(\frac{d_0 - d_t}{d_0}, -1, 1\right) \times \frac{\pi}{2.1}\right) \times \left(1 + \frac{A_t}{5}\right) - \zeta \tag{4.7a}$$

$$A_t = 1 - \frac{|a_e - a_t|}{2\pi} \tag{4.7b}$$

$$R_{2b} = tan\left(clip\left(\frac{d_0 - d_t}{d_0}, -1, 1\right) \times \frac{\pi}{2.1}\right) \times \left(1 + \frac{3A_t}{20}\right) - \zeta \tag{4.8a}$$

$$A_t = int\left(1 - \frac{|a_e - a_t|}{2\pi}\right) \tag{4.8b}$$

Equations 4.7a and 4.8a present two variations of this reward function that differ in reward scaling as a function of the component's orientation relative to that of the expert. d_0 is the distance between the initial position and the expert's position, and similarly, d_t

is the distance computed at time t. The orientation gain, A_t, is a factor that scales of the distance-based reward by up to 20% depending on orientation correctness. It is uniquely defined for R_{2a} by Equation 4.7b and for R_{2b} by Equation 4.8b where a_e is the angle of the expert and a_t is the angle of the component as assigned by the agent at timestep, t.

We add a non-linear function to the normalised distance to aggressively push the agent towards the expert target. This approach removes the burden on the designer to weigh the terms and prevents the agent from finding sneaky ways to maximise the reward without simultaneously achieving the designer's goals. Once the agent starts moving in the right direction, it should get easier to reach the destination. Reward functions 4.7a and 4.8a can be applied to environments with both discrete and continuous action spaces.

4.2.5.4 R_3: Reward based on Agent's Expertise

The reward signal is reformulated a third time, combining ideas from the previous two sections. First, we use wirelength and overlap to encode the reward in a way that is meaningful to the agent. Secondly, we normalise it relative to the initial conditions and best historically known values. Combining these two allows the agent to autonomously learn across layouts without requiring expert knowledge. Thus we surrender control over what makes a good layout and allow the agent to define it as its training necessitates.

Equation 4.9a demonstrates the reward function as a linear combination of HPWL and overlap, where n and m are user-assigned coefficients that respectively weigh the HPWL and overlap. Equation 4.9b defines H_t the normalised HPWL relative to the reset state and expert (best historically known) value h_o and h_e respectively. Equation 4.9c defines the inverted overlap contribution, O_t where, $o_{t,s}$ refers to the normalised overlap of sector s at timestep t. The early termination penalty is retained as with all previous rewards.

$$R_3 = tan\left(\frac{nH_t + mO_t}{n+m} \times \frac{\pi}{2.1}\right) - \zeta \tag{4.9a}$$

$$H_t = clip\left(\frac{h_o - h_t}{h_o - h_e}, -1, 1\right) \tag{4.9b}$$

$$O_t = \left(1 - \frac{1}{s}\sum_s o_{t,s}\right) \tag{4.9c}$$

Furthermore, if the agent finds better valid expert values, they are updated during the reset step at the start of a new episode. The reward signal will therefore change throughout the learning process. This challenges algorithms with a replay buffer since they will be reusing inconsistent training samples, and thus, the training process may not converge or require an extended amount of time. However, its potential lie in removing all

aspects of human bias from the handcrafted training layouts. During its interaction with the environment, the agent will encounter situations where the HPWL is better than the expert. Allowing the agent to update these parameters will result in consistent expert targets. This is important because it will allow generalisation to unseen layouts.

4.2.6 Dataset

We use a single layout from the dataset described in Section 4.1.3, and by choice of moveable components, we create five variations. The central component is an IC with eight pins, while the rest are two-terminal devices consisting of SMD resistors, capacitors and diodes. Based on this layout, two datasets are derived, namely, D_1 and D_2. D_1 contains three unique layouts of increasing complexity. D_2 contains five layouts, each having a particular movable two-terminal device. All layouts in both derived datasets have a single moveable component marked by unlocking it in the KiCad (Bautista et al., 2022) software suite. Appendix B.2 presents pictorial representations of the individual layouts for both datasets. Notice that the pads are generally omitted from our drawings since they are only for aesthetic purposes and may hinder the visual identification of overlapping parts.

Recall that the single-component setup simplifies the problem, and our goal is to evaluate the efficacy of learning a policy. For our purposes, a single circuit with five variations is sufficient as it enables us to determine whether a policy can be learned for a specific scenario and whether this approach can facilitate the learning of generalisation behaviour.

4.2.6.1 Expert Position and Orientation

Reward signals mimicing the human designer, specifically R_{2a} (Equation 4.7a) and R_{2b} (Equation 4.8a), require the expert position and orientation of the moveable component. Similarly reward signals that normalise reward parameters with best historical value, including R_3 (Equation 4.9a) may use the expert setup as an advanced starting point. The `pcb` object stores the layout's original position and orientation values as placed by the expert engineer and can be used for these purposes.

4.2.6.2 Data Augmentation

Resulting from a limited dataset and our desire to maintain a single layout, a data augmentation module is used to scale the limited data to virtually unlimited. Literature suggests (Raileanu et al., 2021) that augmentation may improve training time by requiring fewer rollouts and tends to better generalise to variations of the training data and priorly unseen environments. In our case, before the start of an episode, we augment the original layout

by applying a translation followed by a rotation to the locked portion. The moveable component is still randomly initialised. Any expert target information is altered accordingly.

4.2.7 Training

The training process is parallelised at numerous levels to increase productivity while ensuring we utilise the machines adequately. Moreover, due to numerous distinct experiments, each requiring multiple runs, parts of the process are automated to minimise human error and ensure consistency throughout the research project.

The data used for training is collected through interaction with the environment. Therefore different runs initialised with distinct seed values can lead to amassing different trajectories of varying quality that may or may not lead to learning the desired policy. For this reason, quantitative results increase our confidence and demonstrate the agent's abilities to explore and learn regardless of variations. A single experiment is repeated with different albeit deterministic seed values four to ten times. Concerning Figure 4.12, the number of runs is queued and parallelised over several workers. When the queue is empty, the workers are terminated one by one and averages with standard deviation are calculated.

Individual runs are monitored in Tensorboard (Abadi et al., 2015), an experiment logging tool with a web-based User Interface (UI). It holds information from multiple runs and updates the UIin real time. When all experiments have finished, the generated data is automatically processed, and a `.pdf` report containing charts and tables is automatically generated. An automated reporting system significantly reduced the time spent aggregating results while greatly minimising human error, ensuring consistency in an iterative development process.

4.2.7.1 Choice of Learning Algorithms

On-policy algorithms such as TRPO and PPO collect data using the most recent version of the policy and may struggle to learn on tasks where good trajectories are hard to identify. In particular preliminary experiments with continuous action spaces suggested that learning was highly challenging. The upside is that they tend to converge quicker, so we investigate TRPO and PPO advanced policy optimisation algorithms, mostly with discrete action spaces. We employ TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) hybrid actor-critic methods to address challenging learning environments. Their replay buffer helps retain critical trajectories for learning. Additionally, preliminary experiments showed that TD3 and SAC outperformed competing algorithms regarding performance and stability. They also offer distinctive features for search space exploration that may be desirable for specific tasks.

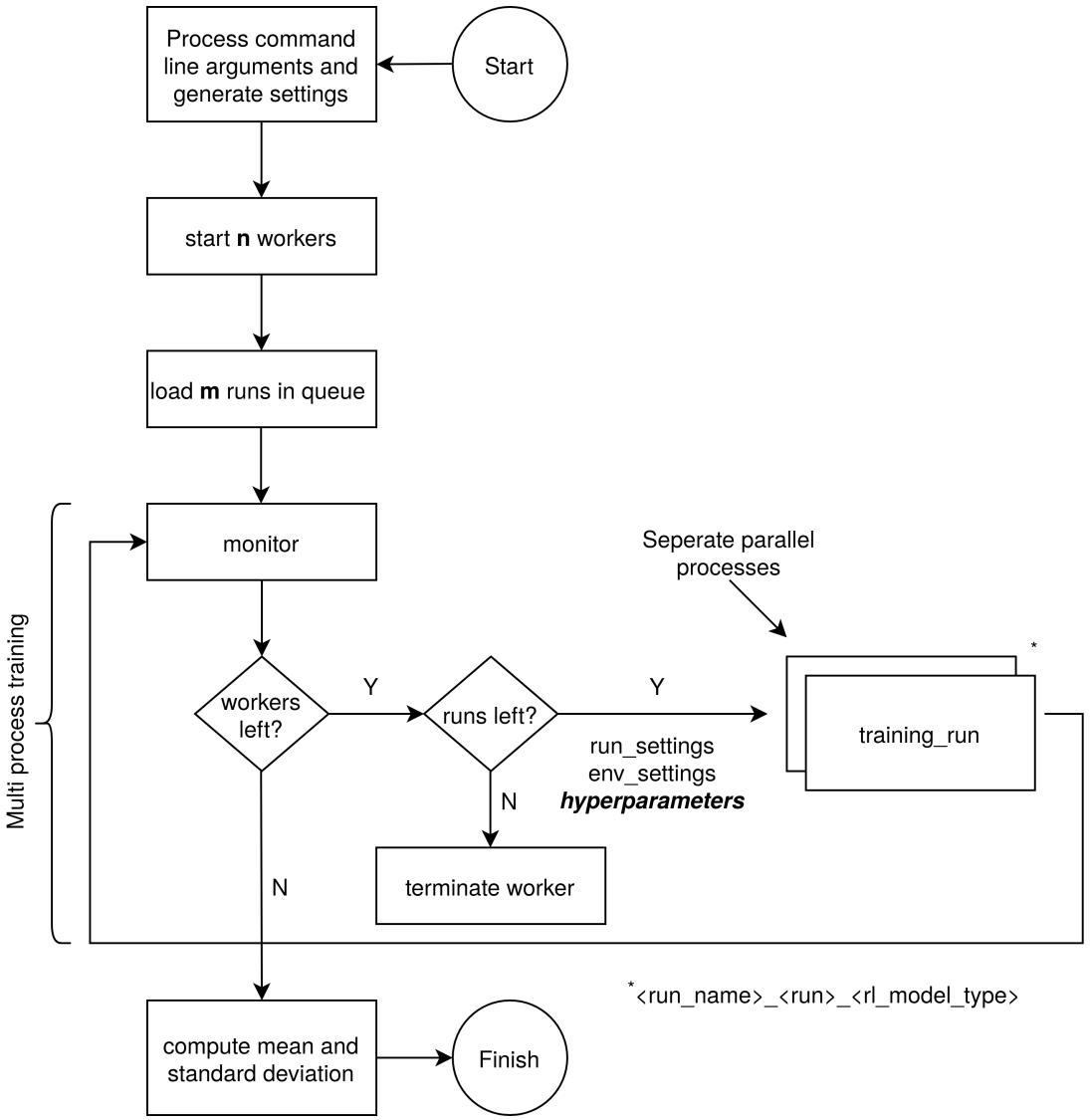


Figure 4.12: Flowchart illustrating parallelised training procedure. n worker processes are spawned to execute the m queued training runs. Idle workers are terminated when no queued runs are available.

Four RL algorithms are used for the experiments in this section. TRPO by Schulman et al. (2015) and PPO by Schulman et al. (2017) are on-policy algorithms while TD3 by Fujimoto et al. (2018) and SAC by Haarnoja et al. (2018) are off-policy algorithms. The default configuration from Stable Baselines3 (Raffin et al., 2021) is summarised in Table 4.9. The framework retains the default parameters as proposed by the original authors.

Hyperparameter	TRPO	PPO	TD3	SAC	Description
Learning rate	1e-3	3e-4	1e-3	3e-4	Learning rate for adam optimiser
Rollout steps	2048	2048	-	-	Rollout steps per environment
Replay buffer size	-	-	1e6	1e6	Size of replay buffer for off-policy algorithms
Batch size	128	64	100	256	Minibatch size for updates
Training frequency	8192	8192	200	1	Weight update frequency in steps
Gradient steps	640	1280	200	1	Gradient steps performed per update
Environemnts	4	4	4	4	Number of simultaneous envrionments
Gamma	0.99	0.99	0.99	0.99	Discount factor
Policy network	32, 32, 128, 64, 64		400, 300		Policy network architecture
Value network	64, 128, 64		-	-	Value network architecture for on-policy algorithms
Q network	-	-	400, 300		Q network architecture for off-policy algorithms

Table 4.9: Default configuration for Stable Baselines3 RL algorithms.

4.2.7.2 Neural Architecture Search

A combined hyperparameter and neural architecture is performed to identify optimised parameter setups and neural topologies that may potentially yield higher performance. With the environment set up according to Table 4.11 and the search space bounded by the constraints in Table 4.10, Optuna (Akiba et al., 2019) is used to carry out the multi-objective optimisation process with a TPE sampler (Bergstra et al., 2013, 2011). This process is repeated for both discrete and continuous action spaces.

Hyperparameter	Value	Description
Neural network layers	1 - 3	Number of neural network layers for policy and Q or value networks
Hidden layer size	16 - 512	Number of neurons in the hidden layers. May be distinctly for each hidden layer
Activation function	ReLU or tanh	Choice of activation function, applied to all neurons in the hidden layers

Table 4.10: Hyperparameter ranges considered for the NAS.

4.2.8 Experimental Setup

This section describes the experiments conducted to study the effectiveness of the MDP formulation. Initially, a proof-of-concept experiment is performed using R_1. Subsequently, we evaluate the efficacy of reward R_{2x} by employing our optimal architecture in conjunction with Stable Baselines3 (Raffin et al., 2021), using both action spaces. Finally, we introduce R_3 (Equation 4.9a), which enables the agent to discover wirelength targets as part of the exploration process autonomously.

Parameter	Value	Description
timesteps	1e6	Number of training timesteps
max_steps	200	The maximum allowable steps per episode
step_size	0.5 | 1.0	The step length for discrete and continious action spaces respectively
reward_type	R_{2a}	Reward function as summarised in Table 4.8
data_augmenter	True	Data augmenter is used for orientation and translation
runs	6	The results will be averaged over six runs

Table 4.11: Default single-component environment parameters.

The RL algorithms and the environment have a significant amount of tunable parameters that may affect the training process in both time, performance or otherwise. The essential parameters are listed in Table 4.11 and will be assumed throughout the rest of this research project where applicable and unless otherwise stated.

4.2.8.1 Experiments with a Discrete Action Space and R_1

TRPO and PPO are used to train policies in our environment guided by R_1 in Equation 4.5b. Scaling constants were identified empirically, taking the following values for HPWL $h_s = 4$, overlap $o_s = 4$ and stepping penalty $S_s = 1$. The scaling limitations of R_1 were known at the design stage, and thus no extensive effort was expended in studying parameter relationships. However, its value lies in demonstrating that a policy can learn placement techniques using the proposed MDP formulation. Two experiments assess the policy's ability to learn placement techniques. The first uses a static environment without data augmentation, while the second introduces variation through data augmentation. The default environment setup in Table 4.11 is assumed, and the RL algorithms are set up according to Table 4.9.

4.2.8.2 Experiments with a Discrete Action Space and R_{2x}

Two distinct experiment sets are proposed on the two datasets in Section 4.2.6. The first uses the three unique layouts in D_1, each with increasing complexity resulting in three separate experiments. The second uses D_2 comprised of five layouts and assesses the ability to learn generalisable placement techniques across distinct layouts and results in a single experiment. In all cases, data augmentation is used to introduce variation.

Before conducting the experiments, a combined hyperparameter and neural architecture search is performed to identify the optimised setup, shown in Table 5.6. RL policies

trained with the optimised setup are presented alongside the default configuration of Stable Baselines3 (Raffin et al., 2021), listed in Table 4.9.

4.2.8.3 Experiments with a Continuous Action Space and R_{2x}

The experiments described in the previous section are repeated using a continuous action space. A hyperparameter search is performed to identify the optimised architecture, shown in Table 5.8 and resulting experiments are presented alongside the default configuration of Stable Baselines3, listed in Table 4.9.

4.2.8.4 Experiments with a Continuous Action Space and R_3

Regarding R_3, we are investigating the impact of a changing reward signal on training performance by examining two different replay buffer sizes: 300k and 600k. These sizes were selected based on preliminary experiments that indicated near-optimal values of expert parameters were achieved after 100k steps. Subsequent experiments investigate whether desirable placement behaviours can be evoked on demand by varying the weighting of the reward signal parameters in Equation 4.9a. Specifically, we will study three configurations: one that equally weighs both HPWL and overlap and two that will emphasise one term at the expense of the other. The choice of figures is arbitrary since our goal is to observe whether distinct behaviours can be elicited and whether they result in improved performance rather than finding the optimal tradeoff. The problem setup and presentation of results are identical to that described in the previous section 4.2.8.3.

The experiments outlined are performed twice, once using the optimised architecture listed in Table 5.8 in Section 5.2.3.1 and another using the Stable Baselines3 (Raffin et al., 2021) default configuration listed in Table 4.9. Evaluations are carried out side-by-side.

Experiment (#)	Replay buffer size	HPWL coefficient	Overlap coefficient
1	300k	1	1
2	300k	1	2
3	300k	2	1
4	600k	1	1
5	600k	1	2
6	600k	2	1

Table 4.12: Combined replay buffer and parameter experiments for R_3.

4.2.8.5 Additional Experiments

Additional experiments were carried out to establish clear baselines and identify the effect of particular parameters on training performance. The former comprises preliminary tests for setup validation. The latter, amongst others, include assessing the impact of episode and step length on training performance and reward maximisation. For additional information, please refer to Appendecies C.1-C.3

4.2.9 Evaluation

Experiments are evaluated based on accumulated reward and their ability to generalise across layouts. We aim to systematically evaluate the fundamental mechanics of orientating a single component in an otherwise fixed circuit. We are not interested in peak training performance but in identifying and understanding the relationships that influence the training process and learned behaviours. Learning across a training set of layouts, not unseen layouts, is very difficult. This is because the dataset is based on human handcrafted layouts and, therefore, is subject to bias and inconsistency. While we will perform training across several layouts, this element will limit the depth of our experiments. That said, our evaluation comprises a thorough hyperparameter optimisation procedure, followed by rigorous empirical experimentation leading to necessary actionable results. Said differently, the conclusions drawn from single-component experiments are mandatory and will provide the basis for the multi-component approach to be discussed in Section 4.3.

4.3 Multi-Component Iterative Placer

This section addresses the fourth objective by designing a multi-component iterative PCB placer capable of learning generalisable placement techniques. It is divided as follows:

1. A highly configurable multi-component training environment is introduced. It extends from the single-component environment presented Section 4.2.2.

2. Describe experimental procedures that thoroughly test the adaptive reward signal via rigorous parameter and ablation tests, the effect of variable-sized replay buffer and accelerated learning through expert knowledge.

3. Describe an evaluation strategy that allows a fair comparison against Simulated Annealing (SA) and presents results without ambiguity and in a standard manner.

4.3.1 Environment

The multi-component environment extends the concepts of the single-component environment to all movable parts of the circuit, effectively addressing the leakage of expert bias through the observation space. It follows a similar flow to an OpenAI Gym (Brockman et al., 2016) environment, implementing the three essential functions (init, reset, and step), but it does not adhere to its specification. The fundamental difference arises during the step method, wherein the policy is invoked for all movable components. Each time, it samples an updated view and acts accordingly. As a result, the number of training data points collected per episode varies for different circuits and equals the product of moveable components and episode steps. In the upcoming subsections, we present a succinct description of the environment and discuss episodic flow in further detail in Section 4.3.3.

4.3.1.1 Observation Space

The observation space is identical to the single-component environment described in Section 4.2.2.1 and summarised by the associated Table 4.5. However, the procedure for deriving the observation changes and is explained in Section 4.3.2.

4.3.1.2 Action Space

Based on the conclusions derived in Section 5.2.6, we have adopted the continuous action space introduced for the single-component environment. The reader is referred to Section 4.2.2.2, where the continuous action space is introduced and accompanied by an associated summary in Table 4.7.

4.3.1.3 Reward

The reward mechanism used to assign the agent credit is built upon R_3 originally proposed in Section 4.2.5.4. Recall that we aim to learn generalisable placement techniques without expert knowledge. Therefore reward needs to be based on problem-related parameters and assign credit indiscriminately across layouts to achieve this goal. Based on these ideas, credit is assigned after each step according to Equation 4.10. It comprises three terms, Euclidean Wirelength (EW) favouring the agent, HPWL benefitting component clusters common to specific nets, and overlap helps maintain a legal (overlap-free) layout. The weighting of these parameters provides a way to control the behaviour learned by policy. For example, emphasising the EW over HPWL promotes the agent to find the best placement with lesser regard to, and often at the expense of increased wirelength for its neighbours. Alternatively, if HPWL is emphasised over EW, the agent will be motivated

to collaborate with neighbouring components to minimise the net length spanning across the component cluster.

$$R_t = tan\left(\frac{nW_t + mH_t + p(1 - O_t)}{n + m + p} \times \frac{\pi}{2.1}\right) - \zeta \tag{4.10}$$

The value for n, m and p scale the individual component but the term inside the parenthsis retain a value of $\frac{\pi}{2.1}^c$ magnitude. The constituent parameters for EW W_t, HPWL H_t and overlap O_t are computed as defined in the following Equations 4.11a-4.11c:

$$W_t = clip\left(\frac{w_0 - w_t}{w_0 - w_e}, -1, 1\right) \tag{4.11a}$$

$$H_t = clip\left(\frac{h_0 - h_t}{h_0 - h_e}, -1, 1\right) \tag{4.11b}$$

$$O_t = \frac{1}{8}\sum_{i=0}^{7} O_{t,i} \tag{4.11c}$$

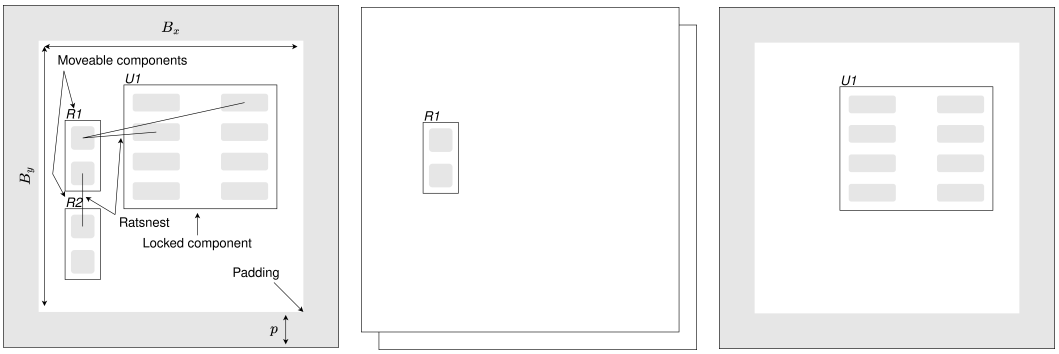
w_0, w_t and w_e are related to the EW, at timestep zero, at timestep t and the expert value respectively. Similarly for h_0, h_t and h_e albeit relating to the HPWL. $O_{t,i}$ is the overlap of segment i at timestep t. W_t and H_t are maintained within a magnitude of one by clipping the excess, while overlap never exceeds unity by design. Doing so prevents their linear combination from exceeding unity and therefore $\frac{\pi}{2.1}^c$ in Equation 4.10.

If better expert parameters are found during an episode, they are updated during the reset step before the commencement of the next episode. Most works in the literature (Badriyah et al., 2017; Holtz et al., 2020; Ismail et al., 2012) use a single wirelength approximation. However, we employ both on the basis that minimising EW will solely benefit the agent while reducing HPWL will benefit the group related by the multi-pin net.

4.3.2 Computing Observations in a Multi-Component Setup

The computation of the observation is a significant change moving from a single to a multi-component environment. The agent still represents a single component, albeit it has access to the entire netlist graph, primarily stored within the environment. Through C++ pointers, the agent can access the most recent version and subsequently compute the appropriate observation in between policy invocations.

The process of computing these metrics is similar to the one outlined in Section 4.2.3.1. We elaborate on the original technique by drawing every movable component on a distinct layer. Line-of-sight and overlap masks are generated based on the current node, and bitwise logical operations are performed between the layer containing the current component and all the remaining layers containing the moveable and fixed components.



(a) Combined layered PCB (b) Moveable components (c) Fixed circuit and padding

Figure 4.13: PCB netlist representation as a stack of images. Leftmost image depicts the entire layout, where each moveable component is assigned a distinct layer as shown in the centre, and the anchor is locked on the right.

This step requires significant computation that increases nonlinearly with the number of components present and will be the bottleneck preventing this approach from scaling to large circuits. However, it suffices for this thesis because we aim for a proof of concept focusing on generalisation to unseen circuits. Figure 4.13(a) illustrates a complete layout containing all the components on the board and padding. Figure 4.13(b) stacks the moveable components on distinct layers. In contrast, Figure 4.13(c) contains the remaining fixed component and padding on an additional single layer. The overlap contribution from all the sectors is normalised, and the maximum value across all the layers is taken. The observation space is identical to that presented in section 4.2.3, and the action space is continuous and conforms with the details of Section 4.2.4.2.

4.3.3 Episodic Flow

The previous section detailed how an observation is generated for a single-component in a multi-component setup. In this section, we will explain how an episode progresses in the context of learning general policies. The agent's role is to perceive the surrounding environment and take actions to co-optimise its location, maximising the reward signal. In contrast, the environment is the host and encompasses all movable and fixed components. Consequently, to allow achieving our objectives, it must be able to accommodate any number of agents and dynamically swap PCB layouts of arbitrary size. To elaborate, when the environment step method is called, it sequentially invokes the policy on each movable component in the netlist, collecting a new observation and taking the appropriate action each time. This sequential process is necessary because the actions taken

by the agent on the previous component may influence the observation captured for the current component. Configuring the problem at both the global level (environment) and local level (agent) is necessary to enable this functionality and switch features throughout the exploration of the problem. Tracking performance and representing complex data in various forms is essential for understanding the implementation's merits and limitations.

The flow diagram in Figure 4.14 provides a high-level overview of a training run. The flowchart on the left depicts the process carried out by the environment. Its primary function is to load a distinct layout at the beginning of each training episode, invoke the agents, and monitor the episode termination condition. On the right side of Figure 4.14, a more detailed depiction of the environment initialisation is provided, including the random layout selection, layout augmentation, and initialisation of the constituent agents.

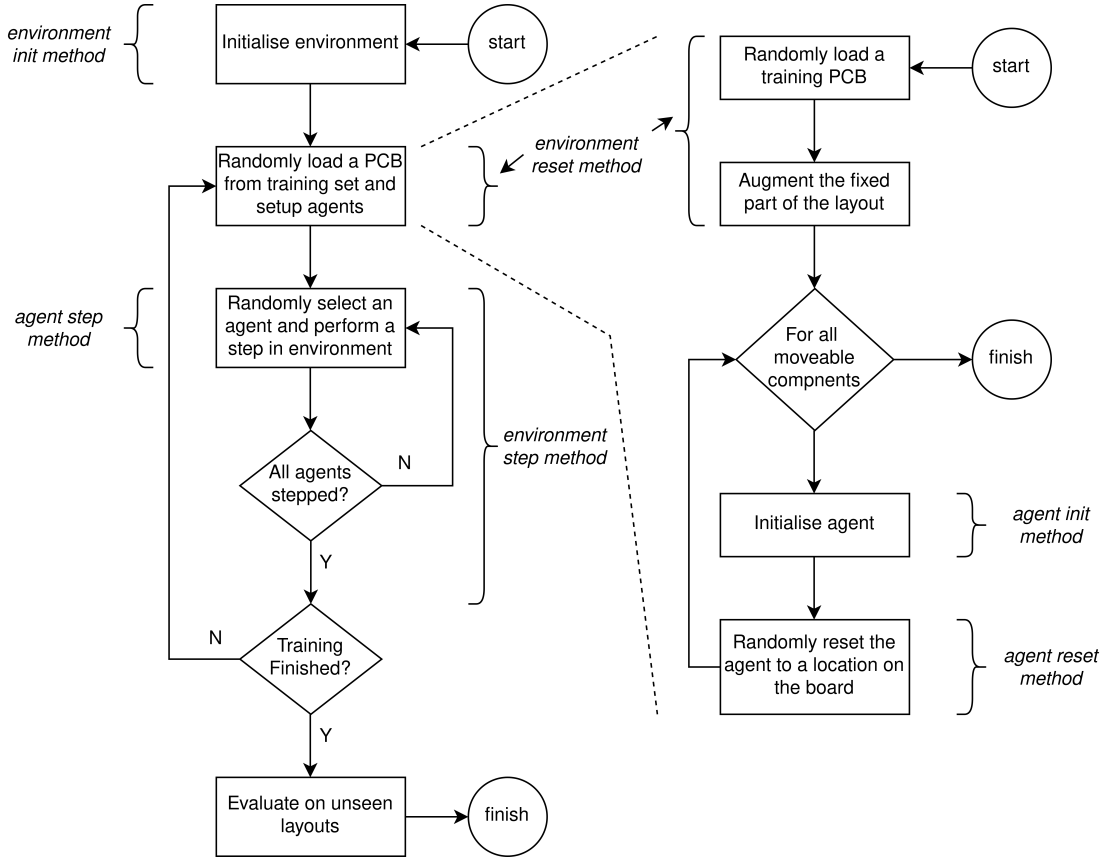


Figure 4.14: Flowchart illustrating the multi-component training procedure. The environment randomly selects a layout and initialises each movable component to a random location. The policy is applied to each component in every step until a terminal state is reached. Following training completion, evaluation using unseen layouts is performed.

4.3.4 Dataset

The dataset for training policies in multi-component setup comprises the same 30 unique circuits described in Section 4.1.4.2. Each circuit contains one fixed component and otherwise moveable components. The fixed component is generally an IC central to the circuit. Intuitively, a feature circuit on the PCB layout comprises multiple subsystems segregated by wires for analogue circuits and buses (e.g. SPI) for digital ones. Locking the central component is based on this observation and serves as an anchor point for the remaining components. In this thesis, we sampled a subset of nine unique circuits, with layout size limited to (20x20mm) and not exceeding 12 components. Six circuits comprise the training dataset M_T, and the remaining comprise the unseen testing dataset M_U. This decision was made in the interest of training time since the multi-component setup is inherently sequential, and the computation required scales non-linearly with increasing layout area and the number of components. Table B.5 enumerates the sampled circuits.

4.3.5 Training

The highest-performing RL algorithms identified from the single component experiments in Section 5.2.5.3, TD3 and SAC, have been adapted to suit a multi-component context.

4.3.5.1 Hybrid Actor-Critic Algorithms

The conclusions drawn from the single component experiments, summarised in 5.2.6, showed that TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) were the best-performing learning algorithms. When judged by an expert designer, the former yielded better circuit layouts, while SAC consistently surpassed TD3 in training performance. The multi-component setup proposed in this chapter cannot use the Stable Baselines3 (Raffin et al., 2021) framework because the environment does not conform to the standard OpenAI Gym (Brockman et al., 2016) specifications and therefore, vanilla implementations of TD3 and SAC were developed as proposed by the original authors. The hyperparameters, policy network and Q network architectures are established in Table 4.9 for TD3 and SAC. All remaining parameters are set to the default values proposed by the original authors, coincidentally corresponding to those set by Stable Baselines3 (Raffin et al., 2021). The custom implementation of both algorithms was verified against Stable Baselines3.

4.3.5.2 Replay Buffer

During training, the agent may encounter better values for EW and HPWL than those assigned by the expert designer, if any. When training without the best historical knowledge

(expert knowledge), the agent has to discover these parameters as part of exploration through trial and error. As these parameters update, the reward signal changes, and the replay buffer's constituents become relatively inconsistent. The inconsistency between the data points is worst during the initial episodes because the agent will frequently find better and better values. For this reason, a custom replay buffer is implemented to allow easy resizing. The replay buffer is initially small, but its capacity gradually increases throughout the training lifecycle. Therefore, the buffer size is small when the data is highly inconsistent as the policy learns better behaviours, and the likelihood of finding drastically better parameter values decreases. As a result of increased data stability, the buffer capacity can be increased accordingly, providing the agent with a larger pool of data points. We assume no knowledge of the expert parameters and allow the learning process to find better values a through the trial-and-error. For practical reasons, we dedicate 5000 steps to identify ballpark values of the expert parameters.

4.3.6 Experimental Setup

The experimental setup investigating iterative multi-component placement is detailed in the upcoming sections. The experiments differ by choice of weighting in the reward function, and we assign various ballpark combinations to study potentially differing performance and behaviours. Identifying optimised configurations (e.g. through Optuna (Akiba et al., 2019)) is left as a task for future work since the aim of this thesis is to investigate end-to-end AI workflows capable of generalisable behaviour. Thus using ballparks figures allows us to study diverse behaviours that may potentially be sub-optimal.

4.3.6.1 Parameter Trade-off Experiments

We investigate the weighting of reward parameters and study how emphasising particular aspects influences learned behaviours. We use four setups, and the configuration is detailed in Table 4.13. The first three tests emphasise a single aspect of the cost function by assigning it a high weighting of 60%, while equally dividing the remaining weight among the other parameters. The fourth setup emphasises wirelength in terms of HPWL and EW by equally assigning them 40% contribution and using 20% for the overlap. The weighting coefficients sum up to ten, ensuring consistency across setups.

4.3.6.2 Ablation Experiments

Ablation tests are conducted to quantify the importance, or lack thereof, of specific reward elements. In this series of tests, we reduce the reward function from three ele-

EW	HPWL	Overlap	Description
6	2	2	Trade-off emphasised on EW
2	6	2	Trade-off emphasised on HPWL
2	2	6	Trade-off emphasised on overlap
4	4	2	Trade-off emphasised on wirelength

Table 4.13: Multi-component parameter experiment configurations.

ments to two and observe the relationships that emerge when using either EW or HPWL in conjunction with overlap. We study three scenarios for each wirelength-overlap configuration. Firstly, we assign an equal weighting of 50% to both parameters, and then we prioritise one over the other by assigning it 80% and the remaining 20% to the other parameter. Table 4.14 summarises the configurations used in the ablation tests.

EW	HPWL	Overlap	Description
0	5	5	Ablation test ignoring EW and equal weighting
0	8	2	Ablation test ignoring EW and favouring HPWL
0	2	8	Ablation test ignoring EW and favouring overlap
5	0	5	Ablation test ignoring HPWL and equal weighting
8	0	2	Ablation test ignoring HPWL and favouring EW
2	0	8	Ablation test ignoring HPWL and favouring overlap

Table 4.14: Multi-component ablation experiment configurations.

4.3.6.3 Additional Experiments

Additional experiments were carried out to understand the impact of particular features on learning and generalisation capabilities. We study the effect prior expert knowledge has on learning performance and whether providing it beforehand aids or hinders exploration of the search space. We also investigate the effects of different replay buffer sizes and resizing strategies. For additional information, please refer to Appendix C.4

4.3.7 Evaluation

The evaluation process uses the unseen dataset, M_U and follows both the rationale and procedure outlined in Section 4.1.6. Concerning our method, we track two layouts with varying overlaps, ranging from 0% to 10%, in increments of 10%. This constraint is relaxed because future work can include a legalisation post-processing task to resolve minor overlaps. It should be noted that PcbRouter (Lin et al., 2020) considers components pin centroids for routing and therefore is impartial to minor overlaps.

5 Results & Discussion

Experimental results delivering the proposed objectives are presented in this Chapter. The constructive placement methodology, including the supervised learning circuit quality estimator, is presented first. Subsequent sections validate our novel iterative placement formulation and systematically investigate the application of RL in a constrained single-component setup. Lastly, from the key findings of the single-component approach, we adapt the problem setup to a multi-component setup, demonstrate the ability to learn fundamental and general placement techniques, and deliver a complete solution for optimising the placement of PCB components. The reader is reminded to refer the Chapter 4 when requiring clarification on the experimental setup.

5.1 Constructive Placer

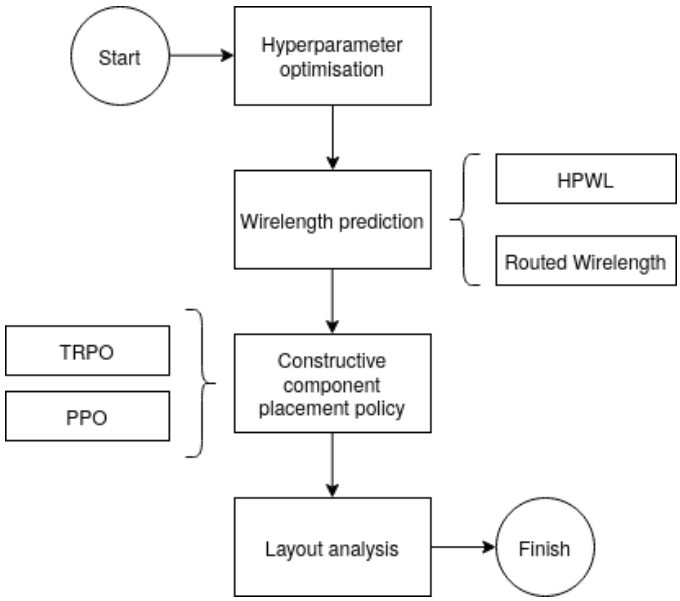


Figure 5.1: Experimental flow for constructive placement methodology.

The experimental results for the constructive placer are presented in this section, whose flow is depicted in Figure 5.1. First, the performance of the graph-level regression task is detailed, including the hyperparameter optimisation results, followed by an analysis of the best setup and prediction performance in terms of HPWL and routed wirelength. Next, the capabilities of learning policies based on using the circuit quality estimator as the state encoder is studied, presenting quantitative measurements and qualitative observations alongside placement by SA (Holtz et al., 2020). The section concludes with a summary of the key findings and a discussion on the efficacy of this approach.

5.1.1 Wirelength Prediction

The wirelength prediction task aims to predict wirelength values directly from a circuit netlist. Since overlap-free wirelength is a measure of circuit quality, the overarching goal is to use such a model as the state encoder for a component placement policy.

5.1.1.1 Neural Architecture Search

The results of a combined hyperparameter and neural architecture search is described in Section 4.1.4.3 and constrained by Table 4.3 are summarised in Table 5.1. The best model prefers Micheli (2009)'s spatial graph convolutional layer to process graph data into a fixed-size embedding. Small batch sizes, deep, fully connected layers, and Rectified Linear Unit (ReLU) activation functions are also preferred. The graph embedding size was identified as the most critical parameter, with a relative importance of 54%. It is followed by 13% for the graph convolution layer's activation function and 9% for the

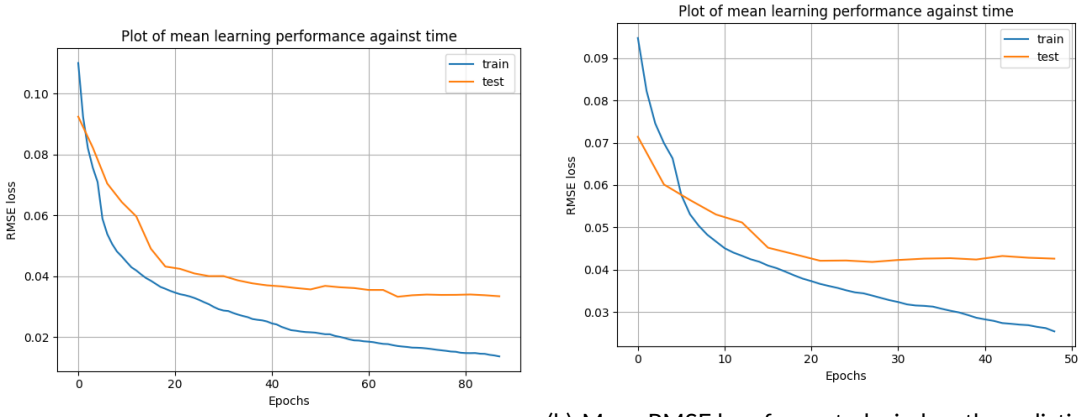
Hyperparameter	**GCN**	**GraphConv**	**GAT**
Learning rate	9.0574e-4	4.5806e-4	8.4668e-4
Batch size	32	32	32
Number of graph convolutional layers	2	3	2
Graph convolutional layer embedding size	89	127	106
Graph convolutional layer activation function	Tanh	ReLU	ReLU
Number of MLP dense layers	4	4	3
MLP layer size	106	112	57
MLP activation function	ReLU	ReLU	Tanh
RMSE (averaged over 4 runs)	11.4015	**7.0754**	12.7678

Table 5.1: Combined hyperparameter and NAS results for HPWL and post-routing wirelength prediction tasks.

batch size. Averaging four runs per trial yielded an optimisation history chart that was robust to outliers evident from clusters of trials centred around a specific objective value.

5.1.1.2 Accuracy

Figures 5.2(a) and 5.2(b) illustrate the training performance for respectively predicting HPWL and routed wirelength targets. In both cases, the average RMSE loss is computed with a window of five samples and is minimised quickly. The model begins to overfit the training data slightly afterwards, evidenced by the divergence between the training and testing loss. However, the early stopping mechanism kicks in, terminating the training process and preventing the unnecessary deterioration of test accuracy. The small RMSE loss value on the y-axis of the plots arises from combining a small batch size of 32 samples with a relatively large training set of $0.7 * 8192 = 5734$ samples. Therefore every epoch $5734/32 = 179$ gradient steps are performed, which are sufficient to significantly reduce the RMSE loss on the first few epochs.



(a) Mean RMSE loss for HPWL prediction

(b) Mean RMSE loss for routed wirelength prediction

Figure 5.2: Training performance for the wirelength predication task.

The model's accuracy is gauged on test data comprised of circuit topologies not seen during training to ensure that it is evaluated under conditions likely experienced in the real world. Table 5.2 enumerates the absolute accuracy recorded over four runs. HPWL prediction achieves an 88.32% accuracy on the test set and 72.08% on the unseen dataset. A notable 16.24% performance degradation on average. On the other hand, predicting the routed wirelength on average yielded an 82.4% accuracy on the test set and 69.49% on the unseen set, a decline of 12.9%.

Analysing the raw data of the best models concerning the models in Table 5.1 has shown that only the graphConv layer (Micheli, 2009) yielded high accuracy on the unseen

	HPWL			Routed Wirelength		
Run (#)	**Test set**	**Unseen set**	**%**	**Test set**	**Unseen set**	**%**
0	87.69%	72.31%	-15.38%	82.01%	68.17%	-13.84%
1	88.50%	70.98%	-17.52%	81.78%	69.53%	-12.25%
2	88.27%	71.36%	-16.91%	**82.92%**	**73.29%**	-9.63%
3	**88.82%**	**73.65%**	-15.17%	82.88%	66.99%	-15.89%
Mean	88.32%	72.08%	-16.24%	82.40%	69.49%	-12.90%

Table 5.2: HPWL and routed wirelength prediction accuracy.

dataset. The graphConv layer exhibits the highest potential for generalising the predictions to unseen circuit netlists. On average, the best trial using GCN for HPWL prediction yielded a test accuracy of 78.63% and an unseen accuracy of 23.86%. Similarly, for GAT, mean test accuracy of 69.03% and unseen accuracy of 15.67% were recorded. In both cases, the deterioration in test accuracy is not surprising due to the higher RMSE error reported in Table 5.1. Interestingly, the discrepancy between accuracy on test and unseen datasets is 58.27% for GCN, 53.36% for GAT and 16.24% for graphConv. A possible cause of GCN's low accuracy is its architecture. However, graphConv and GAT are both spatial GNNs, with the latter being more sophisticated due to the attention mechanism. This leads us to believe that the cause lies in the attention mechanism and may be attributed to failing to learn the correct classification of the relative importance of neighbouring nodes. As detailed by Veličković et al. (2017) and summarised in Section 2.4.1.3, the classification is a function of the node pairs and therefore, including more representative node features may improve generalisation performance.

5.1.2 Constructive Placement

The goal of a constructive placer is to sequentially place components from an ordered circuit netlist onto an empty PCB such that overlap-free wirelength is minimised. Our aim in this section is to encode the problem state space with the predictor developed in the prior section and train an RL agent to perform constructive placement.

5.1.2.1 Learning Policies for Constructive Placement

This section's experiments demonstrate the agent's ability to learn a policy for constructive PCB placement. Figure 5.3 illustrates the training performance of TRPO and PPO, with the former accumulating a higher return while being more consistent across runs, evident from the lower standard deviation across episode runs. Table 5.3 summarises the

results from four runs initialised with distinct seed values and indicates that on average TRPO outperforms PPO by 3.5%.

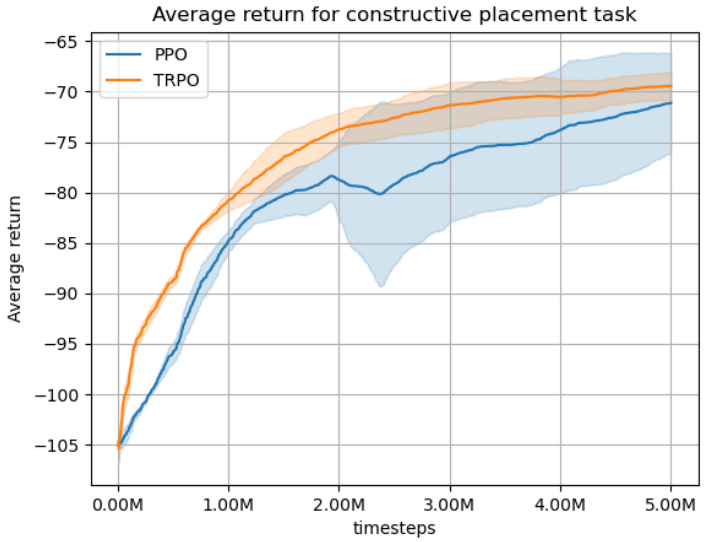


Figure 5.3: Plot of average return for constructive placement training.

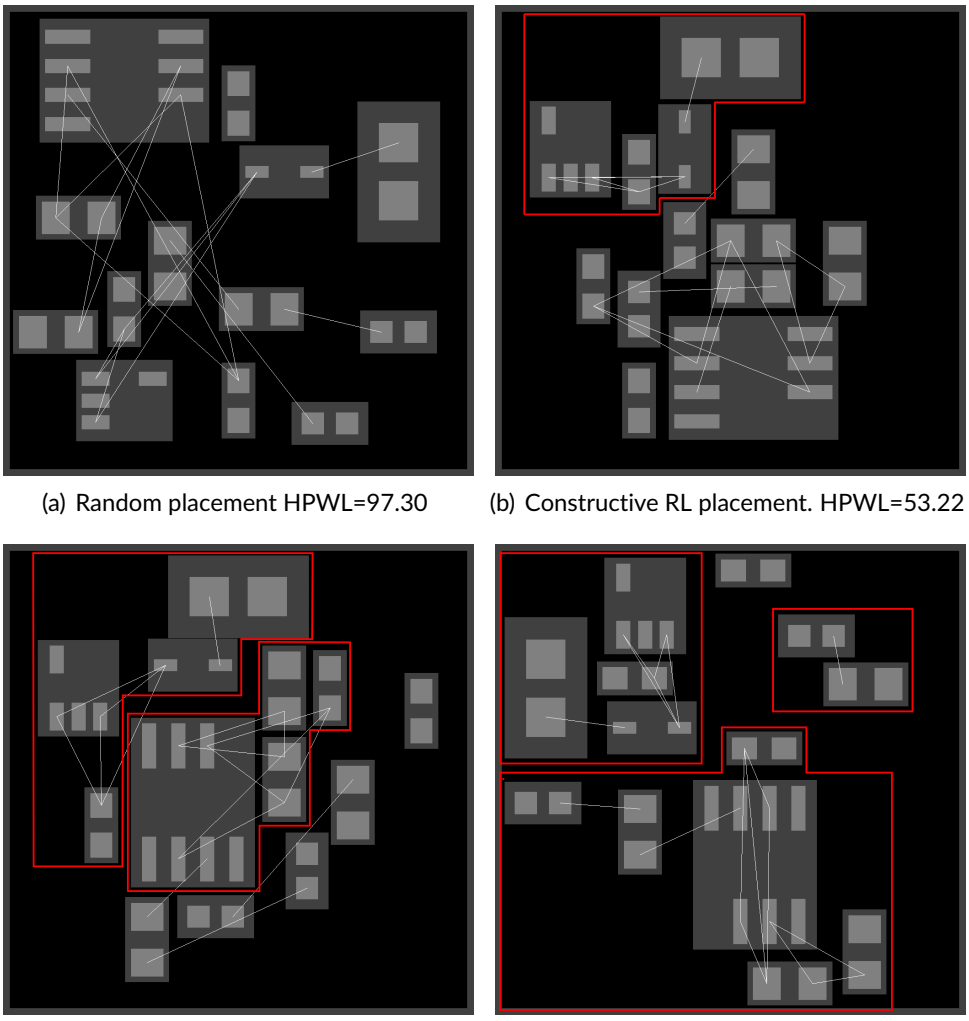
	TRPO	PPO	% TRPO
run #1	-69.7365 ± 0.3140	-70.9859 ± 0.7220	1.76%
run #2	**-68.1976 ± 0.2321**	-68.9814 ± 0.4900	1.14%
run #3	-71.6339 ± 0.4621	-81.4128 ± 1.3659	12.01%
run #4	-70.5304 ± 0.5838	**-68.8251 ± 0.7951**	-2.48%
mean	-70.0246 ± 0.3980	-72.5513 ± 0.8432	3.48%

Table 5.3: Average return after constructive placement training.

5.1.2.2 Inference Performance

The best TRPO policy from the previous section is evaluated against SA-PCB (Holtz et al., 2020) using three distinct layouts, two of which were not seen during training. Table 5.4 summarises the average HPWL and routed wirelength from four evaluations. On average, SA outperforms our approach by 35.04% and 41.65% on the two wirelength metrics.

Figure 5.4 depicts four placement scenarios. Figure 5.4(a) is a random placement, sub Figures 5.4(b) and 5.4(c) are generated by our policy and sub Figure 5.4(d) was generated by SA-PCB (Holtz et al., 2020). An improvement is witnessed between random placement and those generated by our solution. The layouts are compact, and clusters of compo-



(a) Random placement HPWL=97.30

(b) Constructive RL placement. HPWL=53.22

(c) Constructive RL placement. HPWL=60.91

(d) SA placement. HPWL=41.57

Figure 5.4: Placements generated by the proposed methodology alongside SA. The latter demonstrates tightly coupled clusters connected by common nets, a fundamental not witnessed in the proposed method.

nent clusters linked by common nets can be observed within the enclosing red polygons. By contrast, SA does not compact all the components together but is more selective of the components within distinct clusters. Figure 5.4(d) shows that SA is more capable of identifying and orienting components linked by common nets, which is noticeable from the clear demarcation of component groups and ultimately reflected in a lower HPWL.

	Proposed		Simulated annealing		% Simulated annealing	
	HPWL	Routed Wirelength	HPWL	Routed Wirelength	HPWL	Routed Wirelength
Layout 1	64.63	75.98	**43.91**	**46.33**	32.06%	39.02%
Layout 2	56.54	63.20	**34.96**	**36.41**	38.16%	42.39%
Layout 3	60.58	69.59	**39.44**	**41.37**	34.91%	40.55%

Table 5.4: HPWL and post-routing wirelength comparison between RL-based constructive placement and SA.

5.1.3 Key Conclusions for Constructive Placement

Drawing inspiration from the current state-of-the-art AI-assisted floorplanner by Mirhoseini et al. (2021), we designed and evaluated a constructive PCB component placer. A custom OpenAI Gym (Brockman et al., 2016) environment built around a C++ placement engine and an embedded placement quality estimator used as a state encoder for the RL policy. The placement quality estimator was derived from a supervised learning regression task that predicted HPWL with an accuracy of 72% and post-routing wirelength with an accuracy of 69%. A constructive placement policy was successfully trained, albeit falling short when evaluated against SA-PCB (Holtz et al., 2020).

Although the graph level predictions for HPWL yielded a high accuracy on the test set, a deterioration of 16.24% was observed on unseen layouts indicating plenty of room for improvement. Net length estimation tasks for IC physical design (Xie et al., 2021) achieved high accuracy while encoding the netlist graph differently and placing focus on nets rather than components. In the same case, high accuracy was partly attributed to using a customised GAT specific for the task. Altering the netlist representation and including additional attributes can improve accuracy and generalisation performance. Additionally, systematic analysis of the effectiveness of geometrical attributes in conjunction with standard graph convolutional layers (Micheli, 2009; Veličković et al., 2017) for a wirelength prediction task may lead to a better layer architecture. Mirhoseini et al. (2021) note that the graph predictor task used as the state encoder was challenging and proposed a novel edge-based graph convolution, and while they provide a mathematical description of their layer, albeit no further characterisation or performance analysis.

The RL setup currently learns from a single layout. Training across of set of layouts is unlikely to improve performance and suggests that the problem formulation needs improvement. The major limitation lies in the discretised action space, which will prevent the solution from scaling to layout regions that reflect real-world problems. It depends on two conflicting parameters: the size of the layout region and resolution. Our setup requires 400 neurons for a 20x20mm layout region with a resolution of 1mm. PCB layout regions are highly application-dependent and may be small such that they fit within a

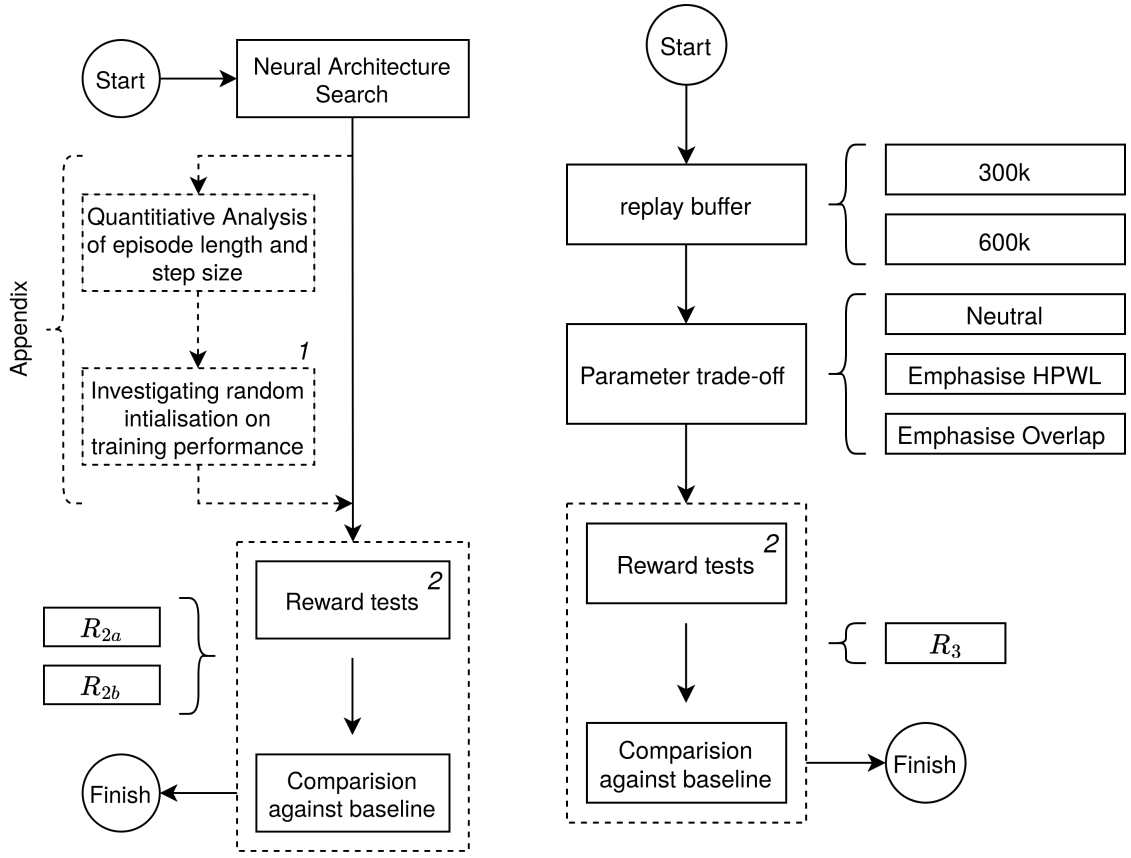
smartwatch or as large as a computer motherboard. Furthermore, component sizes well have a wide dynamic range and the ratio between smallest and largest components is substantial, introducing a size-resolution trade-off. It may span several orders of magnitude, with 10x to 100x typical for small ICs, 1000x for small processors and 10000x for large processors. are commonly found in PCBs of moderate complexity. Mirhoseini et al. (2021) went around this issue by employing a series of deconvolution layers to scale the output of the policy. However, in our case, we believed that the compressed output from the policy would be scaled at the cost of detail, which is highly important in our application.

5.2 Single-Component Iterative Placement

This section presents the results relating to objective two, formulating the iterative PCB placement problem as an RL task and objective three, systematic evaluation of the problem mechanics. We aim to train an RL agent capable of placing a single component in an otherwise fixed circuit. The test results for this section are categorised based on the reward signal:

1. R_1, reward derived from absolute problem-dependent parameters. Proposed in Section 4.2.5.2 and Equation 4.5b
2. R_{2x}, Reward derived from problem-independent parameters guiding the agent to mimic expert designer. Proposed in Section 4.2.5.3 and Equations 4.7a and 4.8a
3. R_3, reward derived from problem parameters conditioned by the agent's expertise for autonomous guidance. Proposed in Section 4.2.5.4 and Equation 4.9a

The experiment flows for each category are illustrated in Figures 5.5(a) - 5.5(b). Briefly, the first category presents a proof-of-concept model verifying that our environment and problem setup can yield the desired behaviour. We immediately move on to the second category, investigating the feasibility of mimicking the expert. We perform a neural architecture search for discrete and continuous action spaces and investigate the learning ability for both reward signal variations in R_2x. Supplementary experiments investigate the impact of random initialisation and episode and step lengths. Finally, we move to the third category (R_3) and assess the ability of an agent to adapt to a changing reward signal and learn relative to it. We achieve this through a combined analysis of replay buffer sizing and different reward parameter weightings.



1 *Applicable for experiments with discrete action spaces*

2 *Three individual layouts and a set of five layouts for assessing generalisation*

(a) Reward signal derived from expert layouts

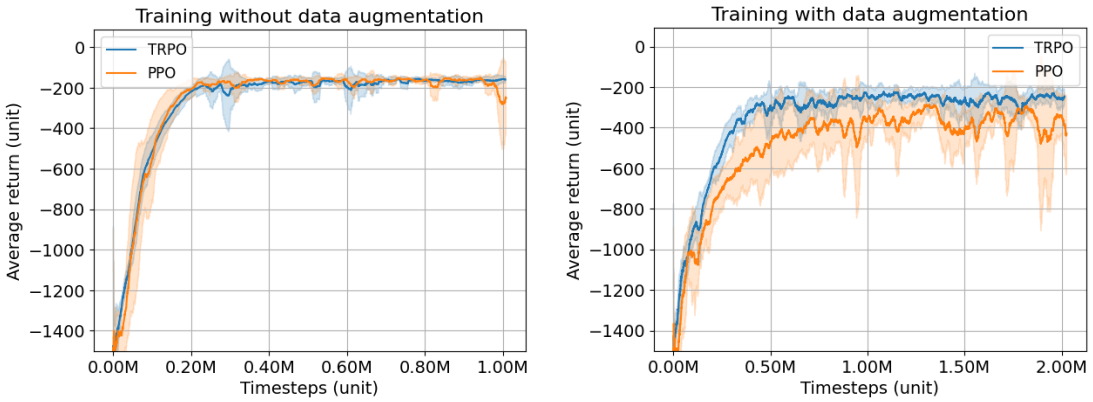
2 *Three individual layouts and a set of five layouts for assessing generalisation*

(b) Reward signal based on agent's experience

Figure 5.5: Experimental flow for iterative single-component methodology. A combined hyperparameter and NAS is followed by experiments assessing reward signals and various features.

5.2.1 Experiments with a Discrete Action Space and R_1

Reward signal R_1 described by Equation 4.5b provides an instantaneous and a terminal reward. Both are important since the first motivates the agent to move in the direction which reduces the HPWL, overlap and steps, while the latter measures the final layout quality. Their combination guides the agent through the episode without losing sight of the end goal: minimal wirelength and no overlap. Figures 5.6(a) and 5.6(b) compare the training performance of TRPO and PPO on a single layout respectively with and without



(a) Training without data augmentation

(b) Training with data augmentation

Figure 5.6: Average return for training on layout D_{1a} guided by reward R_1.

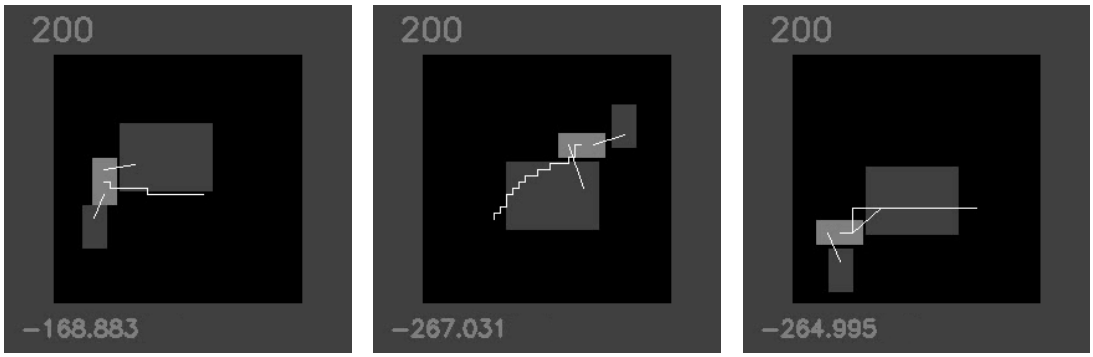
	Training without data augmentation	Training with data augmentation	%
TRPO	-165.60 $\pm$ 85.20	-258.08 $\pm$ 261.49	55.85%
PPO	-181.70 $\pm$ 230.18	-367.70 $\pm$ 380.08	102.36%
% TRPO	9.72%	42.47%	

Table 5.5: Average return for policies trained with reward signal R_1.

data augmentation. Table 5.5 summarises the average return and shows that TRPO is superior to PPO. The gap is especially significant when variations are introduced during training where TRPO attains a 42% higher average return. The higher average return indicates that the policy follows better trajectories and ultimately places the component better than PPO. Furthermore, TRPO tends to be more stable during training, evident from the lower standard deviation and, as a result, a smoother average return curve.

Figure 5.7 depicts the trajectory of policies derived from the experiments mentioned above. The value in the top right corner represents the current step (terminal state), and the negative value in the lower left corner corresponds to the return. Interestingly, the policy finds different ways of placing the component independent of the initial location, as evident from Figures 5.7(b) and 5.7(c). From an expert's perspective, Figure 5.7(a) is the best placement, and Figure 5.7(b) is also acceptable. The latter's return is higher by 33% and is partially attributed to the increased number of steps taken by the policy which highlights the limitations imposed by the discrete action space. While steps are penalised, the HPWL is higher, leading to a lower terminal reward. The agent requires additional steps when the goal region is inclined with an angle that is not a multiple of $\frac{\pi}{2}$. This is

investigated further in Appendix C.2.1.2.



(a) Policy trained with TRPO (b) Policy trained with TRPO (c) Policy trained with PPO

Figure 5.7: Distinct scenarios demonstrating learned placement behaviours under reward R_1.

Throughout our testing, we have found that immediate and terminal elements are necessary to learn a policy. If either was missing, the agent was not motivated to search for a placement and observed the following behaviours. First, when the component is spawned in an isolated region, the agent makes no movement or changes to the orientation and remains frozen. Alternatively, when generated on top of other components, it moves such that all overlap is removed and remains stationary. In both cases, trajectory analysis showed no desire to move towards the goal location. Altering reward parameters to prioritise specific parameters, such as wirelength or overlap, often inhibited the policy from converging, and only after numerous trials a suitable parameter set could be identified. While an optimisation process may identify the optimal parameters, we see this sensitivity as the significant limitation of this approach.

This approach has the merit of serving as a proof of concept that validates our environment and presents a milestone. However, due to intricate reward engineering, to which the training process is sensitive, it is of little value from a practical perspective. As such, no attempts were made to generalise or pursue this method further. With the ground moving under our feet, it is risky and impractical to pursue more complex goals based on this approach. For this reason, in the upcoming section, we reformulate the reward signal to be independent of problem parameters by training policies to mimic the expert designer. This will push the burden of identifying the relative importance of the problem parameters off the designer and onto the policy training process. By guiding the policy with the expert target, the training process is now responsible for implicitly identifying the importance of wirelength and overlap, which should be emergent features.

5.2.2 Experiments with a Discrete Action Space and R_{2x}

A paradigm shift in how we reward the agent is brought about with R_{2x}, where we attempt to learn policies for optimising the placement of circuit components by mimicking the expert. The two variations investigated are respectively described in Equations 4.7a and 4.8a correspondingly termed R_{2a} and R_{2b}. This prevents specialised reward engineering and directly rewards progress intending to learn placement techniques as part of the process. In this section, we present the results of a combined neural architecture and hyperparameter search and then discuss our findings on the efficacy of using R_{2x} with discrete action spaces.

5.2.2.1 Neural Architecture Search

A neural architecture search aims to find the optimal architectural parameters of a neural network. Concerning our work, this means identifying the optimal configuration for the policy and value neural networks including number of layers, corresponding width neurons and activation function. The search is bounded by limits described in Table 4.10. This section describes the optimal architecture for TRPO and PPO learning algorithms with a discrete action space identified after 64 trials. Table 5.6 summarises the optimised architecture and the corresponding objective value.

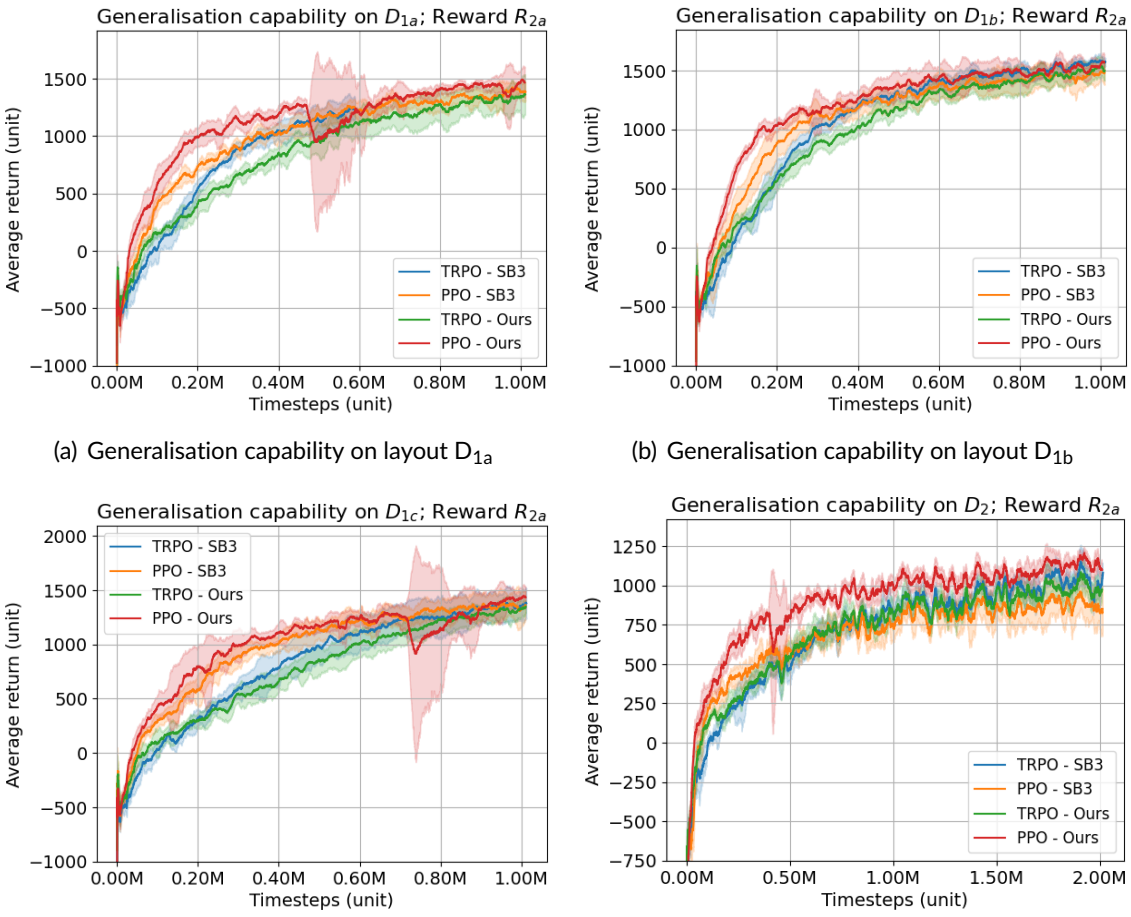
	TRPO	PPO
Policy network	[320, 458]	[377, 511]
Value network	[364]	[433]
Activation function	tanh	ReLU
Best trial value	1821.84	1575.51

Table 5.6: NAS results for policies with discrete action spaces.

5.2.2.2 Generalisation Performance for Reward R_2

Figures 5.8 and 5.9 present policy training with TRPO and PPO guided by reward R_{2a} and R_{2b} respectively. We evaluate our optimised neural architecture alongside the defaults of Stable Baselines3 (Raffin et al., 2021) on the three unique layouts in D_1 and a set of multiple layouts, D_2. All experiments are repeated six times, utilising a deterministic seed value for each run that is consistent across all experiments.

Similar performance results are obtained for both reward signals however, interesting observations can be made. The best results concerning layouts D_{1a} and D_{1b} are obtained from reward signal R_{2b}, while R_{2a} dominates on layout D_{1c} and when training with D_2 over



(a) Generalisation capability on layout D_{1a}

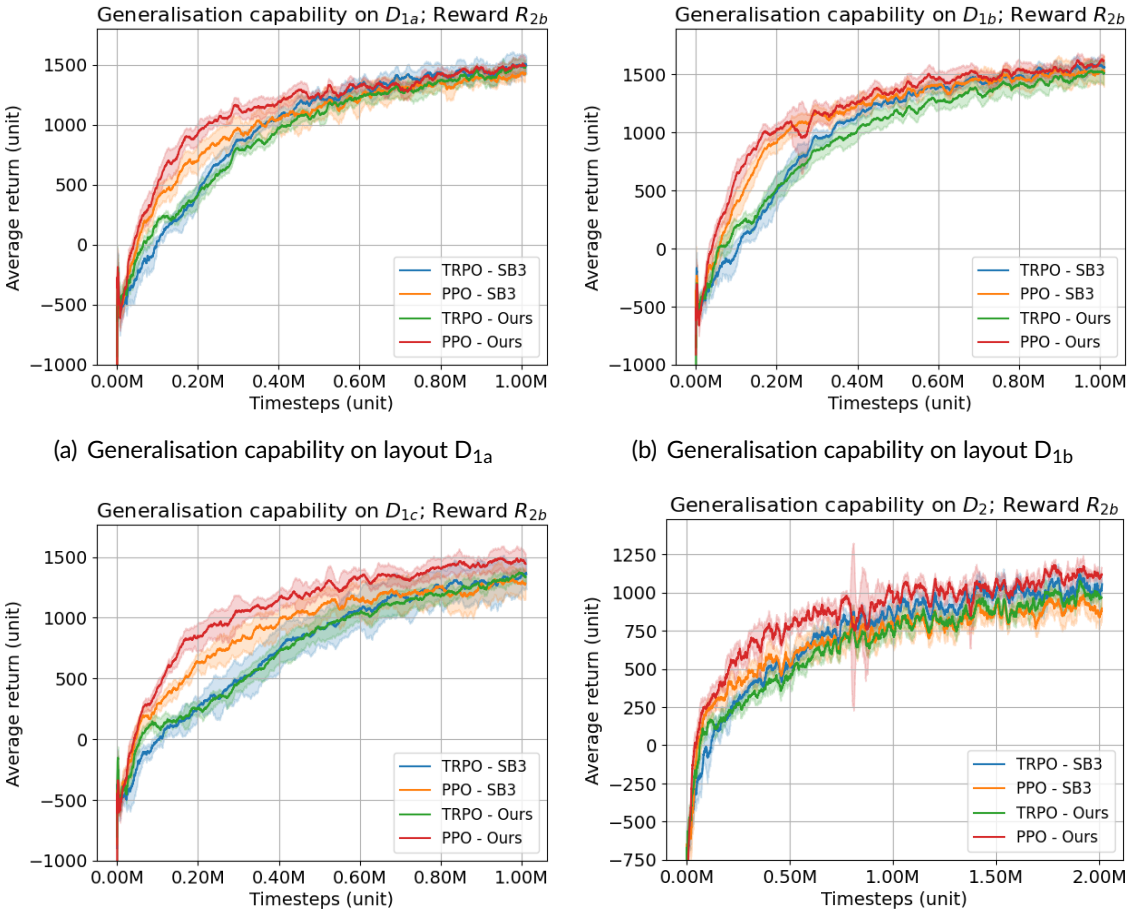
(b) Generalisation capability on layout D_{1b}

(c) Generalisation capability on layout D_{1c}

(d) Generalisation capability on layout D_2

Figure 5.8: Generalisation capability with R_{2a} and a discrete action space. The average return is maximised in all cases, with a noticeable decline when training across multiple circuits.

multiple layout. Recall from section 4.2.6 that layout D_{1c} is relatively complex, containing a moveable component with two multiple-pin nets. Yielding the proper orientation may be more challenging due to the human's pre-defined orientation being sub-optimal or inconsistent with the remainder of the layouts. Furthermore, reward R_{2a} scales the base reward value according to the orientation's relative difference to the target. In contrast, reward function R_{2b} does not scale the base reward value unless the orientation corresponds to the target. Thus if the orientation is incorrect, the latter will lose a 15% reward scaling while the former is still being scaled, albeit by a lower amount. As a result, if the policy cannot identify the correct orientation and constantly alters the value, reward R_{2a} is more likely to lead to a higher return since an offset will result due to reward scaling.



(a) Generalisation capability on layout D_{1a}

(b) Generalisation capability on layout D_{1b}

(c) Generalisation capability on layout D_{1c}

(d) Generalisation capability on layout D_2

Figure 5.9: Generalisation capability with R_{2b} and a discrete action space. The average return is maximised in all cases, with a noticeable decline when training across multiple circuits.

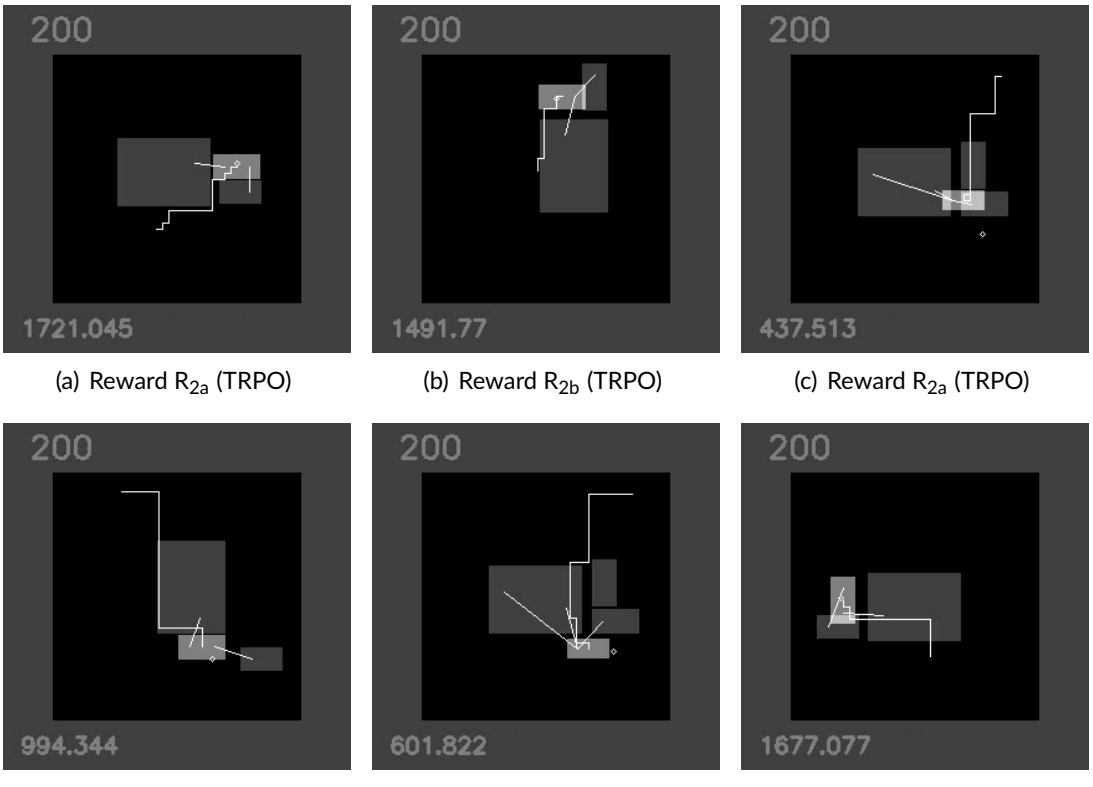
Table 5.7 shows a notable increase in the standard deviation on layout four and more significantly when training across layouts. Since the values presented are obtained from six runs, the cause for the increased variation may be either attributed to sensitivity in the initial conditions or variations across the individual episodes.

When comparing the best policies trained on D_2, we observed no notable difference between reward signals R_{2a} and R_{2b}. Figure 5.10 illustrates trials selectively chosen to highlight the successes and limitations exhibited by the best policies. The policies were selected according to the emphasised experiments in Table 5.7, using PPO with our optimised architecture and TRPO with the default Stable Baselines3 architecture for both reward signals. Each trial performed 200 steps and followed the trajectory marked by the

		Reward R_{2a}			Reward R_{2b}		
	Layout	Proposed	SB3	% Proposed (mean)	Proposed	SB3	% Proposed (mean)
TRPO	D_{1a}	1348.48 ± 381.93	1396.88 ± 365.17	-3.59	1373.84 ± 347.73	**1402.00 ± 357.68**	-2.05
	D_{1b}	1401.22 ± 401.04	1480.74 ± 401.83	-5.68	1376.16 ± 358.36	**1489.19 ± 357.10**	-8.21
	D_{1c}	1281.97 ± 416.13	**1364.62 ± 350.28**	-6.45	1122.63 ± 479.98	1308.73 ± 418.07	-16.58
	D_2	920.48 ± 603.73	**997.00 ± 644.21**	-6.14	901.25 ± 591.23	957.06 ± 604.04	-6.19
PPO	D_{1a}	1437.51 ± 350.44	1413.48 ± 406.77	1.67	**1491.69 ± 392.94**	1345.52 ± 450.51	9.80
	D_{1b}	1478.98 ± 362.64	1518.27 ± 409.84	-2.66	**1531.77 ± 369.11**	1478.27 ± 416.40	3.49
	D_{1c}	**1478.69 ± 412.99**	1285.56 ± 492.40	13.06	1396.78 ± 636.85	1223.09 ± 464.67	12.44
	D_2	***1019.24 ± 575.87***	952.42 ± 582.16	6.56	989.75 ± 645.45	885.97 ± 597.16	10.49

Table 5.7: Average return comparing the optimised models against the established defaults in Stable Baselines3. Training process guided by R_2a.

white line originating from the centre of the moveable component, drawn with high intensity. The miniature circle is the target, corresponding to the placement location assigned by the human designer. The step number associated with the depiction is presented in the top left corner, while the return is noted in the bottom left corner.



(a) Reward R_{2a} (TRPO) (b) Reward R_{2b} (TRPO) (c) Reward R_{2a} (TRPO)

(d) Reward R_{2a} (PPO) (e) Reward R_{2b} (PPO) (f) Reward R_{2a} (PPO)

Figure 5.10: Distinct terminal states demonstrating success and limitations of policies with discrete action spaces.

Often multiple ways exist to place a single component in an otherwise fixed circuit, which is problematic in this experiment. Generally, the agent can move the component to the desired region, often settling with a good orientation. For example, the orientation in Figure 5.10(b) is not deemed correct during training because the human engineer oriented the component a further 90 degrees clockwise. However, it is still a good placement, not considering the small overlap present. Figures 5.10(a) and 5.10(b) illustrate correct placement corresponding with that of the expert. In contrast, Figure 5.10(f) has good positioning and an incorrect orientation leading to a 14.3% overlap.

The average return across these trials drastically varies based on accurate positioning. This is the manifestation of the aggressive tangent function in the reward signal. It also leads to a large discrepancy in return for minimal variation from the target and does not accurately promote the fundamental behaviours necessary for mastering the task. It is exacerbated further by inconsistencies in the layouts arising from manual placement that mislead the agent and inhibit generalisation. The agent is provided with mixed signals, leading to settling somewhere in between. For example, Figures 5.10(d) and 5.10(e) are deemed good placements by an expert but not Figure 5.10(f). The former two accumulate a small return while the latter gathers a significantly higher value, highlighting shortcomings of the engineered reward signal.

Observing the policy behaviour on layouts used in training revealed some inconsistencies. Figure 5.10(c) shows a policy that fails to locate and orient a component. The way it fails indicates that the policy has yet to learn the importance of overlap and that it has not been able to learn the fundamental principles of this task. Recall that reward signal R_{2a} and R_{2b} provided by Equations 4.7a and 4.8a guide the reinforcement learning agent to mimic human-generated layouts. Therefore it is only concerned with the result, while we assume that during the interactive trial-and-error process, it collects data that allows it to learn the fundamental principles of the task - minimise wirelength without overlap. On the one hand, when we observe variations in component orientation that are still deemed correct by the expert, it gives us hope that the agent is learning the basics. On the other hand, when the policy dramatically fails, we lose confidence in its learned capabilities. Failure to train across multiple layouts, evident from the low return and high deviation as well as by observing the policy behaviour, indicates that this method fails to generalise. The possible causes for this are discussed after assessing the same process, albeit with a continuous action space.

5.2.3 Experiments with a Continuous Action Space and R_{2x}

In this section, we repeat the same experiments, assessing reward signals R_{2x} from the previous section, albeit with a continuous action space. Four RL algorithms are considered, these being the on-policy TRPO and PPO and the off-policy TD3 and SAC. First, we present an experiment to identify the optimal episode length, albeit omitting the step size, since, with a continuous action space, the policy controls the step magnitude (and direction) continuously, therefore having granular control. Following is a neural architecture search that identifies optimal architecture for the policy and value or Q neural networks for on-policy and off-policy algorithms, respectively. Finally, we analyse the policy performance concerning reward signals R_{2a} and R_{2b} for generalisation capabilities.

In this section, we repeat the same experiments for assessing reward signals R_{2x} albeit with a continuous action space. Four RL algorithms are considered, these being the on-policy TRPO and PPO and the off-policy TD3 and SAC. Once again, we present the optimised hyperparameters and neural architecture search on-policy and off-policy algorithms, respectively. Afterwards, we analyse the policy performance concerning reward signals R_{2a} and R_{2b} for generalisation capabilities. Supplementary experiments in Appendix C.3.2 investigate the optimal episode length, albeit omitting the step size, since, with a continuous action space, the policy has granular control over the step size.

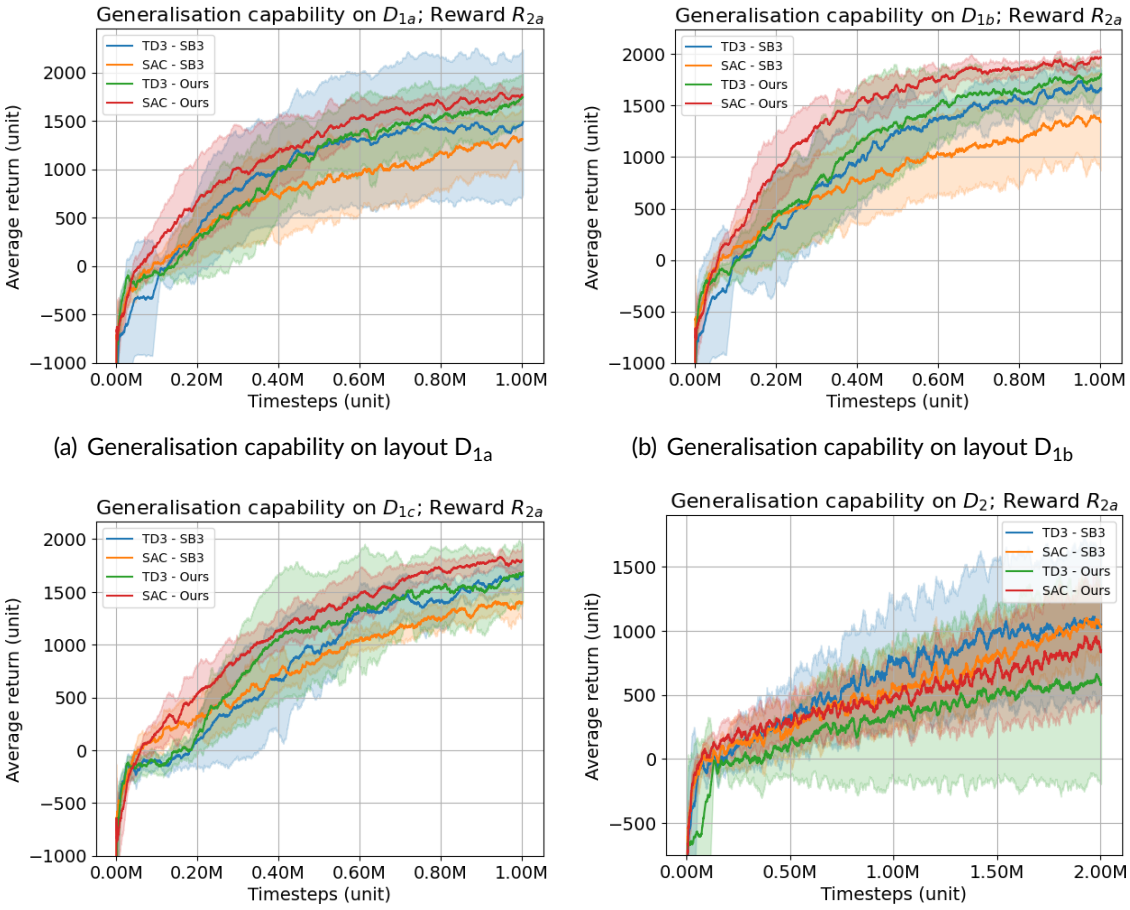
5.2.3.1 Neural Architecture Search

The neural architecture search is performed once more concerning policies with a continuous action space. The boundaries enforced on the search process are noted in Table 4.10 while the resulting topologies are summarised in Table 5.8 along with the objective value. Concerning TD3, the search process leads to an architecture requiring two to three policy layers and favouring two Q-function layers. High importance is placed on the choice of the activation function. In contrast, SAC is more flexible in terms of activation function, albeit recognises high importance on the number of Q-function layers. In fact, an objective value exceeding 1600 was only possible with two layers.

	TRPO	PPO	TD3	SAC
Policy network	[87, 341]	[274]	[237, 356]	[20, 280]
Value network	[237, 346, 414]	[53]	[290, 449]	[126, 501]
Activation function	tanh	tanh	ReLU	tanh
Best trial value	118.260	64.33	2124.08	1993.66

Table 5.8: NAS results for policies with continuous action spaces.

A preliminary test in Appendix C.3.1 showed that TRPO and PPO fail to learn a policy and discuss possible causes. Additionally, we focused on optimising the neural architecture and a further attempt to learn policies could be made by altering parameters related to the learning process. These include the learning rate, the number of observations collected every rollout, and the number of network weight adjustments (gradient steps) per epoch performed following the rollout phase. Based on these results, upcoming tests concerning continuous action spaces are limited to TD3 and SAC.



(a) Generalisation capability on layout D_{1a}

(b) Generalisation capability on layout D_{1b}

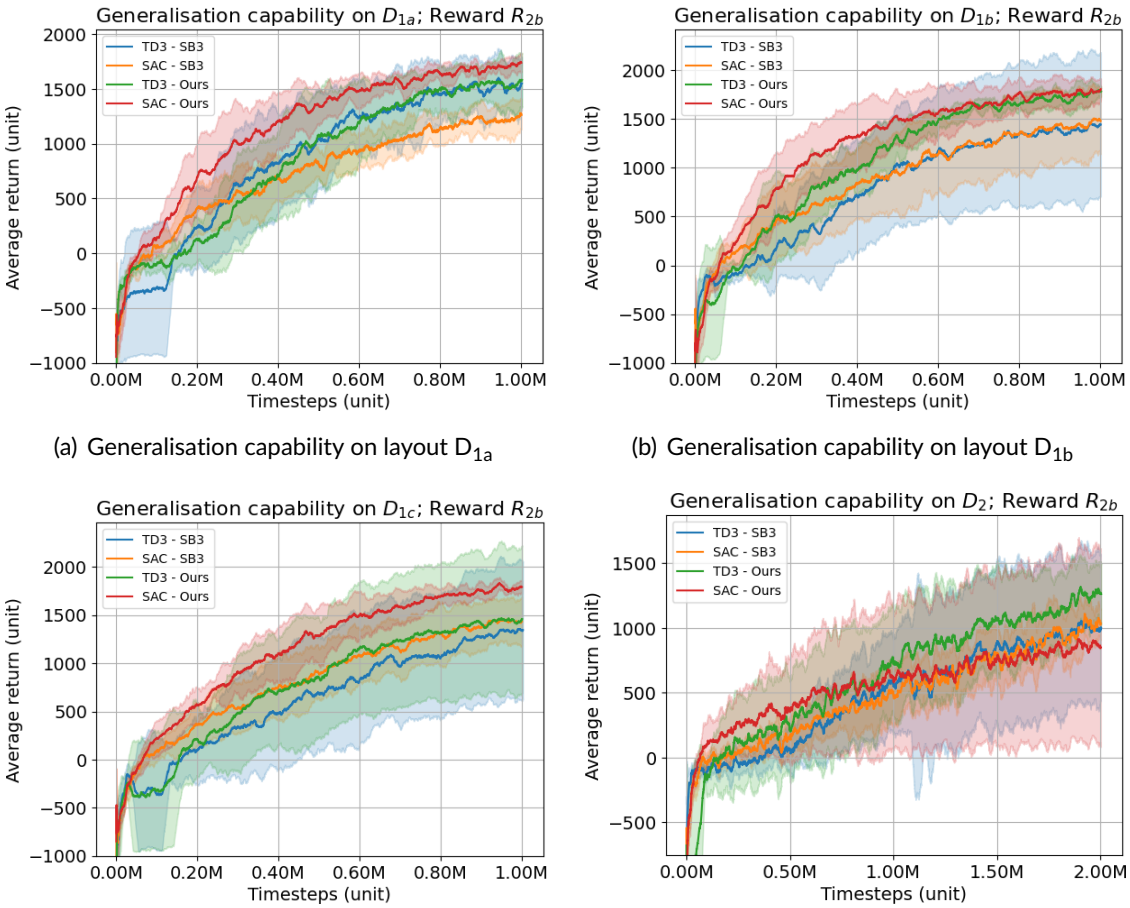
(c) Generalisation capability on layout D_{1c}

(d) Generalisation capability on D_2

Figure 5.11: Generalisation capability with R_{2a} and a continuous action space. When training across multiple circuits, a decline in average return accompanied by increased variance.

5.2.3.2 Generalisation Performance for Reward R_{2x}

The performance of the policy is evaluated numerically in terms of average return and by analysing the exhibited behaviour on different layouts. According to Table 5.9, our optimised TD3 on average outperforms Stable Baselines3 (Raffin et al., 2021) default architecture on all individual layouts in D_1 for both reward signals. The margins are even more significant for SAC, indicating that the optimisation process was even more beneficial. However, this is not the fundamental reason for the increased margins. Some individual runs in TD3 failed to learn a policy, significantly influencing the average due to low average return in the respective runs. One possible reason for the failed policy is the



(a) Generalisation capability on layout D_{1a}

(b) Generalisation capability on layout D_{1b}

(c) Generalisation capability on layout D_{1c}

(d) Generalisation capability on D_2

Figure 5.12: Generalisation capability with R_{2b} and a continuous action space. When training across multiple circuits, a decline in average return accompanied by increased variance.

		Reward R_{2a}			Reward R_{2b}		
	Layout	Proposed	SB3	% Proposed	Proposed	SB3	% Proposed
TD3	D_{1a}	**1617.06 ± 523.56**	1252.18 ± 554.30	29.14	1527.31 ± 495.17	1509.84 ± 575.22	10.35
	D_{1b}	**1729.65 ± 439.62**	1637.48 ± 542.03	5.63	1724.16 ± 415.90	1384.09 ± 444.63	24.57
	D_{1c}	**1571.46 ± 572.24**	1565.79 ± 460.58	0.36	1409.28 ± 414.83	1270.92 ± 466.63	10.89
	D_2	571.87 ± 474.31	1026.89 ± 545.80	-44.31	**1174.29 ± 567.89**	942.97 ± 550.27	24.53
SAC	D_{1a}	**1721.48 ± 387.54**	1248.44 ± 560.80	37.89	1688.31 ± 392.46	1205.56 ± 567.54	40.04
	D_{1b}	**1905.84 ± 367.02**	1312.99 ± 576.53	45.15	1759.81 ± 387.97	1418.03 ± 440.71	24.10
	D_{1c}	**1764.36 ± 367.67**	1349.47 ± 556.07	30.74	1751.07 ± 371.71	1395.81 ± 413.02	25.45
	D_2	808.26 ± 663.29	**977.65 ± 554.6**	-17.33	827.51 ± 531.49	939.30 ± 526.28	-11.9

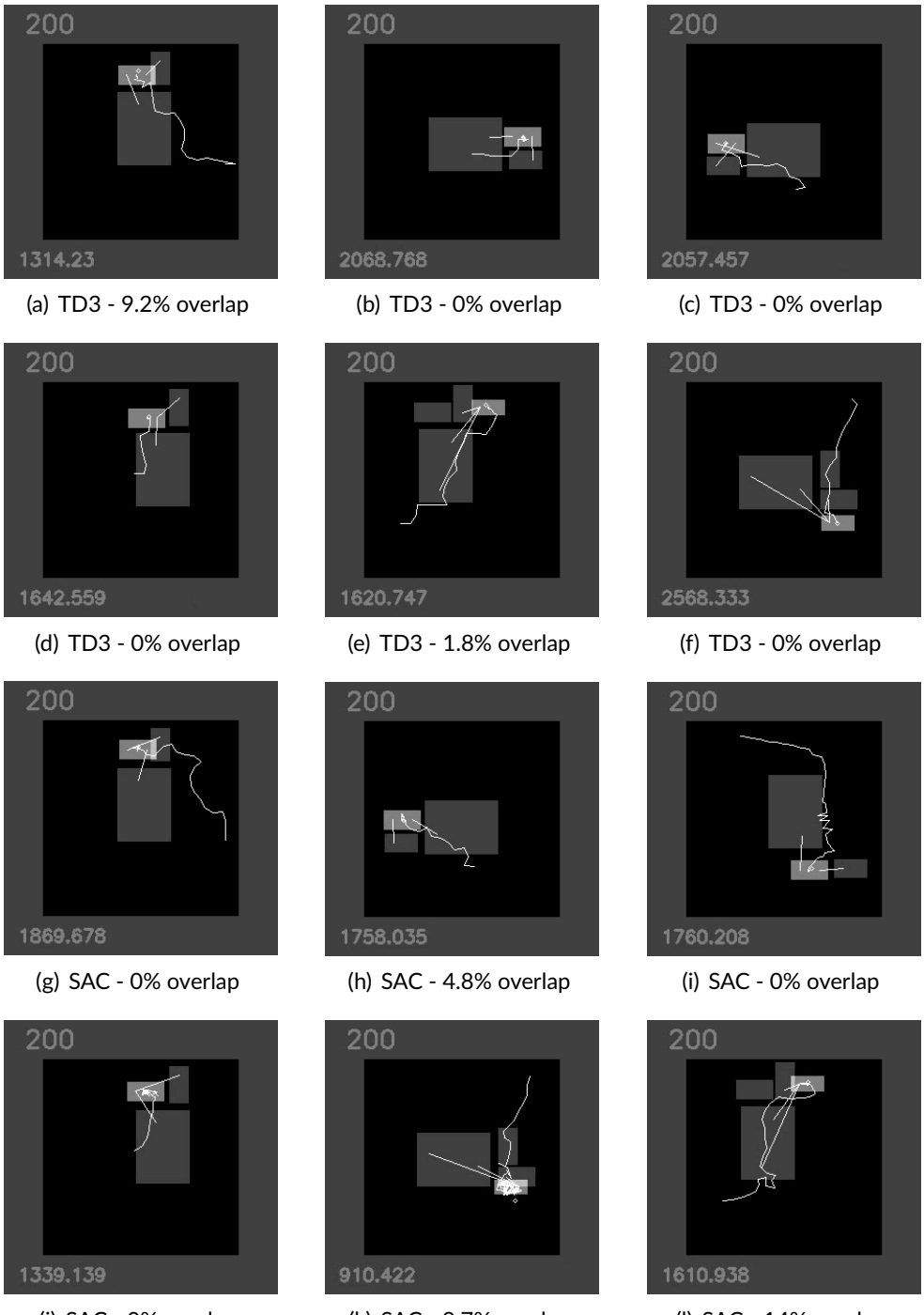
Table 5.9: Average return comparing the optimised models against the established defaults in Stable Baselines3. Training process guided by R_{2b}.

initial conditions that start the process from an unfavourable position. Furthermore, SAC has the upper hand in exploration since it optimises a surrogate function that simultaneously maximises the reward and search space exploration. Therefore SAC may be more effective despite starting from an unfavourable position in the search space.

When learning across multiple layouts with dataset D_2, mixed results are observed, and 75% of the time, the default Stable Baselines3 architecture outperforms our setup. However, regardless of the neural topology, the mean return is relatively poor and accompanied by a significantly higher standard deviation. These numbers do not indicate that either policy can learn across a distinct layout set, thus questioning the ability to generalise to unseen designs. While these numbers are poor across the board, our setup for the experiment concerning multiple layouts is at a disadvantage because the neural architecture search was performed on a single layout, D_{1a}.

Figure 5.11 illustrates the training performance guided by reward signal R_{2a}, while Figure 5.12 depicts the same process, albeit guided by reward signal R_{2b}. Experiments in Table 5.9 show that our optimised architecture was, on average superior to Stable Baselines3 on all unique layouts in D_1 by a margin of 13.49% and 33.89% for TD3 and SAC, respectively. Performance deteriorated when training on D_2 by 9.89% and 14.62% for our optimised TD3 and SAC. Concerning the choice of reward signal, both algorithms tend to perform better with reward signal R_{2a} when training with individual layouts, which is likely attributed to the averaging effect of the reward scaling mechanism.

Figures 5.13(a)-5.13(f) and 5.13(g)-5.13(l) illustrate the behaviour exhibited by the best policies according to the average in Table 5.9. That is, the former is generated by TD3 trained with reward signal R_{2b}, and the latter is generated with a policy trained with SAC and reward signal R_{2a}. The episode step number is located at the top left corner, while the return is noted at the bottom left. Interestingly we observed that the learned policy performs well across layout sets for both algorithms. However, we sometimes noted situations where the agent fails to position the component close to the expected target



(a) TD3 - 9.2% overlap (b) TD3 - 0% overlap (c) TD3 - 0% overlap

(d) TD3 - 0% overlap (e) TD3 - 1.8% overlap (f) TD3 - 0% overlap

(g) SAC - 0% overlap (h) SAC - 4.8% overlap (i) SAC - 0% overlap

(j) SAC - 0% overlap (k) SAC - 9.7% overlap (l) SAC - 14% overlap

Figure 5.13: Distinct terminal states highlighting placement behaviours exhibited by policies trained with TD3 and SAC.

region, as shown by Figure 5.13(k). While SAC outperformed TD3 in average return, the latter resulted in smoother trajectories and better placement, as can be observed from Figures 5.13(a), 5.13(d), 5.13(h) and 5.13(j). Furthermore, in the latter two, it is also evident that SAC tends to oscillate around the target region where TD3 does not. These behaviour patterns may be attributed to the stochastic policy employed by SAC. Albeit subjective and inferred from a limited sample set, the difference in return associated with different episode runs is less noticeable with continuous action space.

5.2.4 Action Space Comparison on R_{2x}

Table 5.10 summarises the best results from the previous generalisation results for discrete and continuous action spaces. Concerning a discrete action space, PPO with an optimised neural architecture is the clear winner outperforming TRPO in all layouts. For a continuous action space, SAC's performance in individual layouts is excellent, only falling behind TD3 on layout D_{1a} by a mere 0.17%. TD3 with a reward R_{2b} seems to be the better algorithm for training across multiple PCB layouts. In all scenarios using an off-policy algorithm with a continuous action lead to an 18.63% higher return on average. Thus we conclude that the additional complexity resulting from a continuous action space is beneficial as it leads to more accurate policies. This is also evident from the trajectory analysis based on Figure 5.10 for a discrete action space and 5.13 for a continuous action space.

Layout	Best Discrete	Algorithm	Reward	Best Continuous	Algorithm	Reward	% Continuous
D_{1a}	1491.69 $\pm$ 392.94	PPO - Ours	R_{2b}	1729.65 $\pm$ 415.90	TD3 - Ours	R_{2a}	15.58%
D_{1b}	1531.77 $\pm$ 369.11	PPO - Ours	R_{2b}	1905.84 $\pm$ 367.02	SAC - Ours	R_{2a}	24.42%
D_{1c}	1478.69 $\pm$ 412.99	PPO - Ours	R_{2a}	1764.36 $\pm$ 367.67	SAC - Ours	R_{2a}	19.32%
D_2	1019.24 $\pm$ 575.87	PPO - Ours	R_{2a}	1174.29 $\pm$ 545.80	TD3 - SB3	R_{2b}	15.21%

Table 5.10: Comparison between discrete and continuous action spaces.

5.2.5 Experiments with a Continuous Action Space and R_3

The section presents the results where policies are rewarded using problem-dependent parameters with the aim of learning generalisable behaviours. In contrast with R_{2x}, the reward signal does not credit the agent for mimicking the expert. Instead, it promotes the agent to autonomously identify the optimal parameters and learn relative to them. Training is restricted to TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) with a continuous action space since they were identified to be the best but retain a relative comparison between our optimised architecture and Stable Baselines3 (Raffin et al., 2021). First, the investigation of the replay buffer's effect on the training performance

is presented. This is done in conjunction with training several policies trading off HPWL and overlap, for which we analyse the exhibited behaviour. Finally, we performed a generalisation performance test identical to those of the previous section, albeit using R_3.

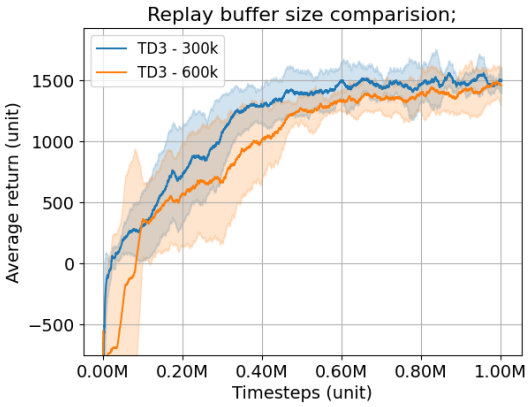
5.2.5.1 Investigation of the Replay Buffer Size on Training Performance

The replay buffer plays a significant role when using reward R_3 (Equation 4.9a) because the HPWL term in the reward function is normalised with the best-known historical value of the parameter. The latter changes as the policy improves and learns to find better placements. As a result, the data in the replay buffer will initially be inconsistent but should stabilise as the policy becomes better. This experiment investigates the effect of the replay buffer size on the training performance. Figure 5.14 depicts several runs, each showing a side-by-side comparison of training performance using a replay buffer size of 300k and 600k samples. Sub Figures 5.14(a) through 5.14(c) utilise TD3 for learning while 5.14(d) to 5.14(f) are generated by policies trained with SAC. For each algorithm, the reward is set up in three ways that prioritise particular parameters in the reward signal. Sub Figures 5.14(a) and 5.14(d) emphasise HPWL, 5.14(b) and 5.14(e) emphasise overlap and 5.14(c) and 5.14(f) equally weight wirelength and overlap. The returns are summarised in Table 5.11, emphasising a comparison between replay buffer size and choice of the learning algorithm. The returns from experiment runs with differing configurations are not directly comparable and would require an alternative evaluation method.

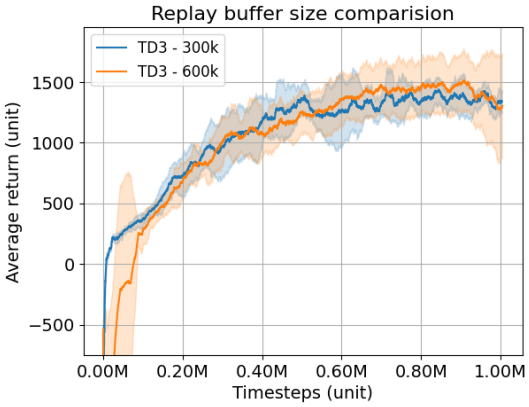
On average, TD3 is less sensitive to the buffer size, evident from the 1% difference in average return. Contrastingly, SAC prefers a smaller buffer size indicative from the 10% increase in average return. Furthermore, in all tests, SAC outperformed TD3 with by a significant 25% on average when using the smaller buffer. Additionally SAC is more stable throughout training evident from the relatively lower standard deviation.

Replay buffer size	300k			600k		
Training Algorithm	TD3	SAC	% SAC	TD3	SAC	% SAC
HPWL = 1.0 Overlap = 1.0	1324.48 $\pm$ 517.16	1765.24 $\pm$ 345.90	+33.28%	1363.72 $\pm$ 520.03	1580.76 $\pm$ 386.99	+15.92%
HPWL = 1.0 Overlap = 2.0	1361.99 $\pm$ 546.16	1647.48 $\pm$ 440.63	+21.05%	1402.37 $\pm$ 508.69	1544.74 $\pm$ 456.96	+10.15%
HPWL = 2.0 Overlap = 1.0	1495.77 $\pm$ 462.27	1846.03 $\pm$ 281.41	+23.42%	1416.54 $\pm$ 402.82	1551.62 $\pm$ 317.45	+9.54%

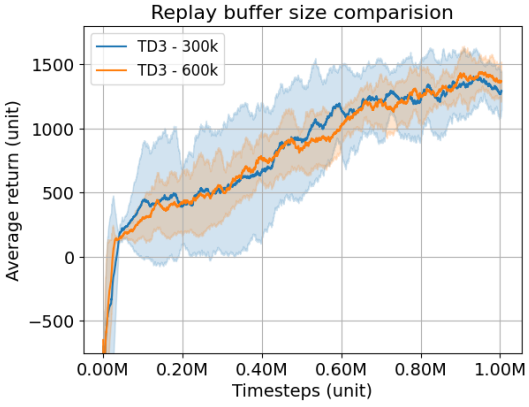
Table 5.11: Average return for experiments with parametrised R_3 configurations. Two replay buffer sizes are considered.



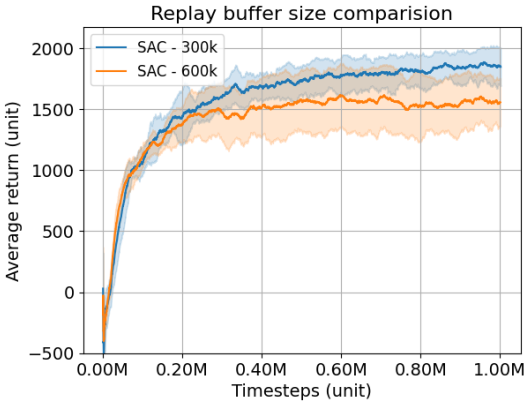
(a) (HPWL=2, Overlap=1) - TD3



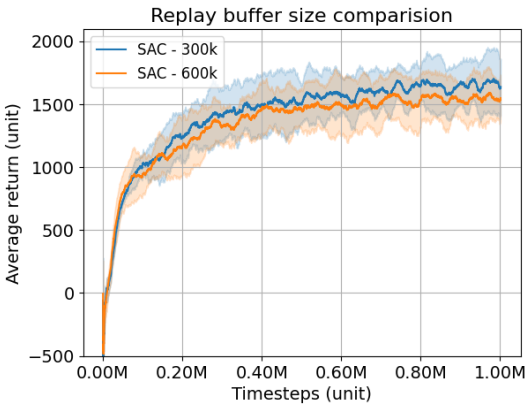
(b) (HPWL=1, Overlap=2) - TD3



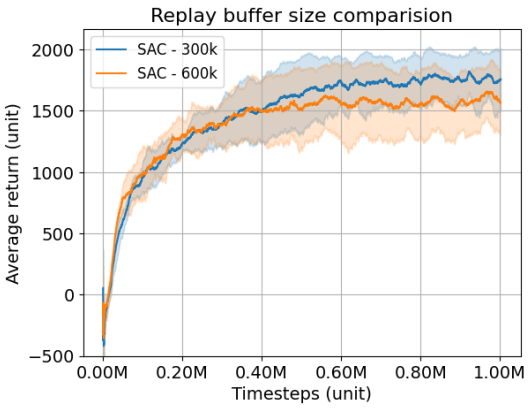
(c) (HPWL=1, Overlap=1) - TD3



(d) (HPWL=2, Overlap=1) - SAC



(e) (HPWL=1, Overlap=2) - SAC



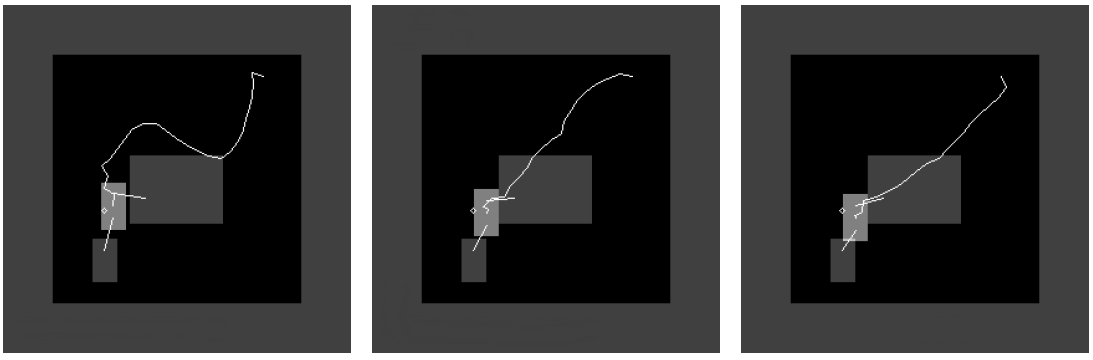
(f) (HPWL=1, Overlap=1) - SAC

Figure 5.14: Average return for training policies with differing configurations of R_3 and replay buffer sizes.

5.2.5.2 Qualitative Analysis of Learned Behaviour

The reward signal comprises an HPWL and overlap terms, and favouring one at the expense of the other will encourage the policy to exhibit different behaviours. Figure 5.15 demonstrates an example of each policy trained in the previous section. Each row in Figure 5.15 illustrates three scenarios corresponding to three policies, from left to right: emphasise overlap, neutral and emphasise wirelength. Sub Figures 5.15(a) through 5.15(c) in the top row were trained with TD3 while 5.15(d) through 5.15(f) in the bottom row learned with SAC. The low-intensity projections (dark) represent the circuit's locked portion and padding. The policy controls the single component drawn with higher intensity (bright), and the white line originating from its centroid is the trajectory taken from a random location. The small circle corresponds to the placement location of the human designer and was used as the target in previous experiments guided by reward R_{2x}.

Interestingly, Figures 5.15(a) and 5.15(d) show that the policy avoids overlap to the extent that it follows a longer trajectory and circles around the locked components or



(a) HPWL=1, Overlap=2 - TD3 (b) HPWL=1, Overlap=1 - TD3 (c) HPWL=2, Overlap=1 - TD3

(d) HPWL=1, Overlap=2 - SAC (e) HPWL=1, Overlap=1 - SAC (f) HPWL=2, Overlap=1 - SAC

Figure 5.15: Illustration of distinctive policy behaviour exhibited by TD3 and SAC when emphasising particular parameters in reward signal R_3.

even alters its trajectory to avoid overlapping. In both cases, the final placement is excellent because it minimises wirelength while avoiding overlap. By contrast, Figures 5.15(c) and 5.15(f) cut straight through to the goal region without considering the overlap in the process. In other words, the policy follows the trajectory that minimises the HPWL contribution and is coherent with the behaviour promoted by the reward function since a higher cost is associated with HPWL and a lower cost with overlap. Interestingly in both scenarios, the final positioning has minimal overlap suggesting that the policy differentiates between the intermediate process of moving closer to the optimal region and the final placement state. Figures 5.15(b) and 5.15(e) are somewhere in between, indicative of the balance between both reward parameters, albeit some overlap is still present in the final state. In practice, the behaviour associated with sub Figures 5.15(c) and 5.15(f) is desired, however all scenarios yield similar results. Some overlap in the final state is not a big issue because it can be removed with a post-processing legalisation step. The scenarios depicted by Figure 5.15 demonstrate that the agent embodies the behaviour promoted by the designer through the reward signal.

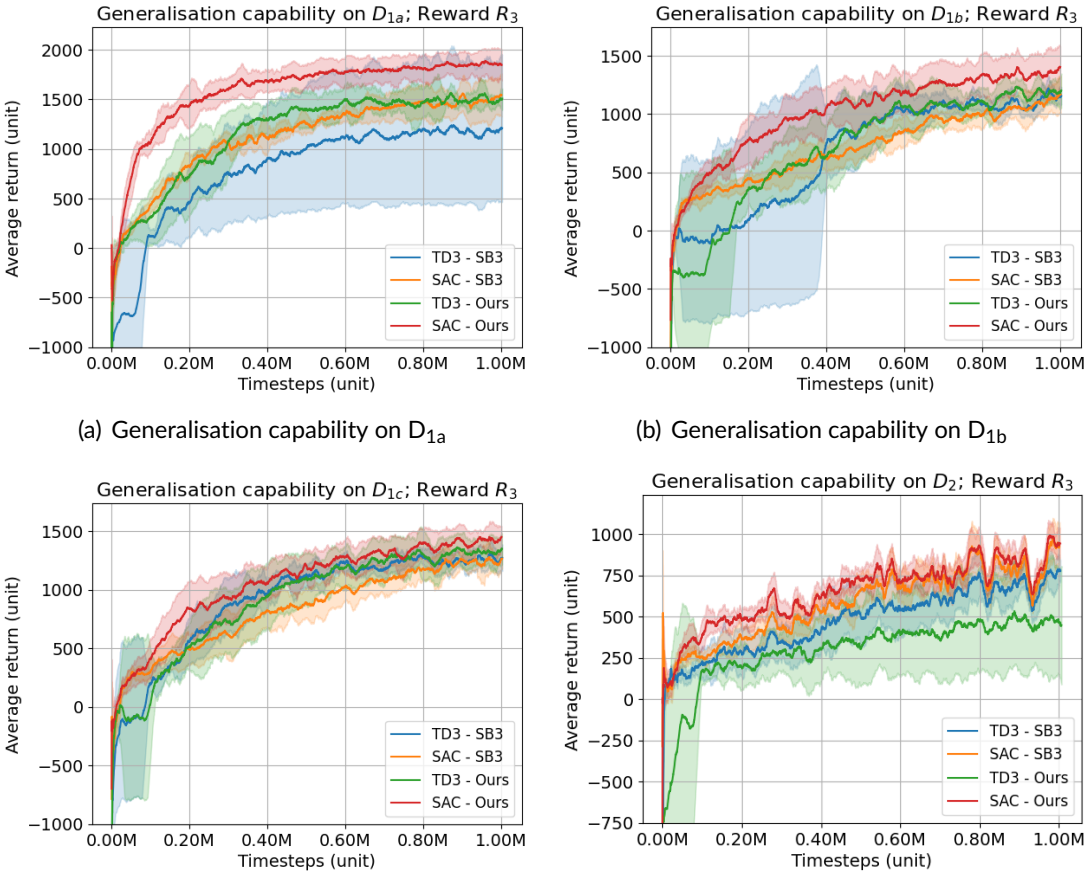
5.2.5.3 Generalisation Performance for Reward R_3

This section assesses the ability of the setup to learn and generalise across different layout sets. Figure 5.16 depicts the plots of average return against the absolute number of steps carried in the environment. Figures 5.16(a) through 5.16(c) train on a single layout with increasing complexity and aim to quantify learning ability. In contrast, sub Figure 5.16(d) uses a set of different layouts and aims to quantify generalisation ability. In all cases, we evaluate our optimised architecture alongside that of Stable Baselines3 (Raffin et al., 2021) and average all experiments over six runs.

Table 5.12 summarises the return for all experiments. Our optimised architecture outperforms Stable Baselines3 on all individual layouts with 11.93% for TD3 and 21.59% for SAC. The return significantly drops when learning across layout sets in D_2 regardless of the algorithm used, with optimised TD3 being outperformed by 32.59% and SAC retain-

Layout	TD3 - SB3	TD3 - Proposed	% Proposed	SAC - SB3	SAC -Proposed	% Proposed
D_{1a}	1170.95 $\pm$ 421.85	1495.09 $\pm$ 462.27	27.74%	1476.09 $\pm$ 364.86	1846.03 $\pm$ 281.41	25.06%
D_{1b}	1141.55 $\pm$ 481.68	1176.18 $\pm$ 457.03	3.03%	1065.43 $\pm$ 438.78	1336.46 $\pm$ 397.88	25.44%
D_{1c}	1249.18 $\pm$ 469.15	1311.96 $\pm$ 421.25	5.03%	1226.96 $\pm$ 395.16	1402.03 $\pm$ 367.46	14.27%
D_2	687.46 $\pm$ 557.45	463.41 $\pm$ 477.89	-32.59%	792.59 $\pm$ 516.42	839.53 $\pm$ 520.34	5.92%

Table 5.12: Average return comparing the optimised models against the established defaults in Stable Baselines3. Training process guided by R_3 with (HPWL=2; Overlap=1) configuration.



(a) Generalisation capability on D_{1a}

(b) Generalisation capability on D_{1b}

(c) Generalisation capability on D_{1c}

(d) Generalisation capability on D_2

Figure 5.16: Generalisation capability with R_3, small replay buffer of size 300k, and a continuous action space. When training across multiple circuits, a decline in average return is accompanied by increased variance. Reward signal configuration (HPWL=2; Overlap=1).

ing the upper hand albeit with a significantly smaller margin of 5.92%. A large discrepancy is noted in TD3 for individual layouts in D_1. A possible cause is the usage of layout D_{1a} during the architecture search and is exacerbated since the RL agents were credited with R_{2a}. Thus some performance loss was expected. The failure to generalise is discussed in the context of the single-component strategy in the upcoming section.

5.2.6 Key Conclusions for Single-Component Iterative Placement

This section presented our single-component, iterative placement environment with discrete and continuous action spaces. We investigated four reward signals and the effect

of multiple elements on learning performance, including choice of action space, episode length, and replay buffer size in scenarios with an adaptive reward signal. In all scenarios, we identified the optimal neural architecture, corresponding training algorithm and rigorously trained policies with increasing difficulty to evaluate the capability to learn fundamental placement behaviours and generalise across layouts. Relating to the second objective of formulating the PCB problem as an RL task, this section showed sufficient evidence (Section 5.2.5.2) of learning policies for orienting a single component in an otherwise fixed circuit. Concerning the third objective of systematically designing and evaluating the fundamental mechanisms of a single-component iterative PCB placement environment. Empirically we found that an episode length of 200 steps was adequate, and the random component initialisation had a negligible impact on the return albeit increased standard deviation. Furthermore, a continuous action space slightly mitigated the random initialisation issue due to increased freedom of movement and reduced the error incurred due to a quantised step. TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) algorithms yielded the best performance and outperformed a discrete action space by 17.67%. Performing a neural architecture search on average resulted in a 23.7% higher return when compared to Stable Baselines3 (Raffin et al., 2021) on individual layouts. The performance deteriorated by 12.26% when learning across layouts, albeit this may be attributed to the optimisation process focused on a single layout.

Reward signal R_{2x} showed that rewarding the agent to mimic the expert yielded good results on individual layouts but did not convincingly demonstrate an ability to generalise. Despite training across layouts, inconsistent behaviour was occasionally observed. This likely arises from the inherent differences of handcrafted layouts because, with particular exceptions, engineers rarely think in terms of HPWL. These reasons suggest that both the goal and the dataset are inconsistent, as evident from Figures 5.10 and 5.13 and the large discrepancy in return. On the other hand, R_3 was proposed based on overlap and wirelength that adapts to the agent's expertise. It partly overcame the inconsistency resulting from R_{2x} and, through iterative self-improvement, allowed the agent to autonomously learn a policy for accurately orienting itself in the circuit. While similar generalisation conclusions were drawn, the behaviour exhibited by the agent was consistent according to the parameters emphasised by the designer, as evident from Figure 5.15.

The takeaway from this approach is that layouts generated by human designers are inconsistent, and the agent cannot learn general techniques due to receiving mixed signals during training. After proposing R_3, results suggest that the limitation lies in the locked portion of the layout, leading to conflicting observations for the agent when training across layouts. In conclusion, we demonstrated that our PCB placement formulation allowed us to successfully train numerous RL policies under differing conditions fulfilling

the second objective. We also presented a thorough investigation of the fundamental mechanisms for single-component placement, thereby also fulfilling the third objective.

5.3 Multi-Component Iterative Placement

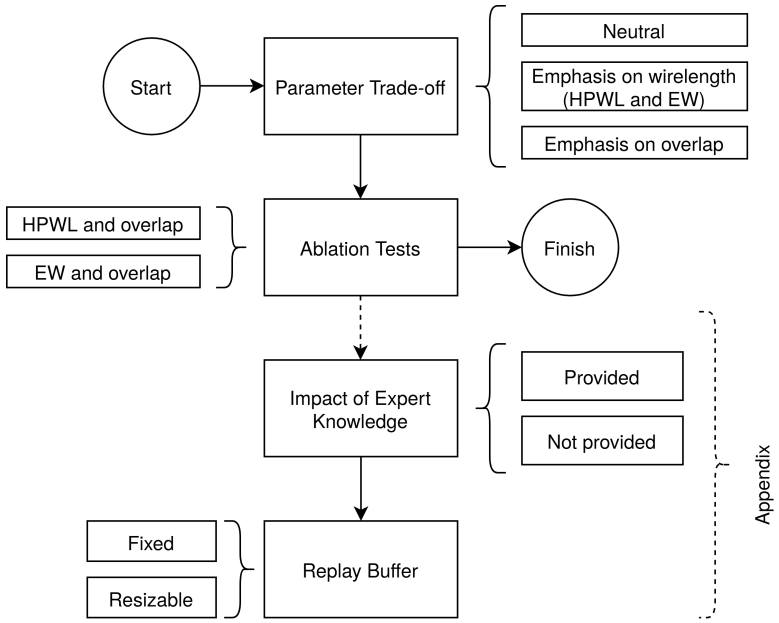
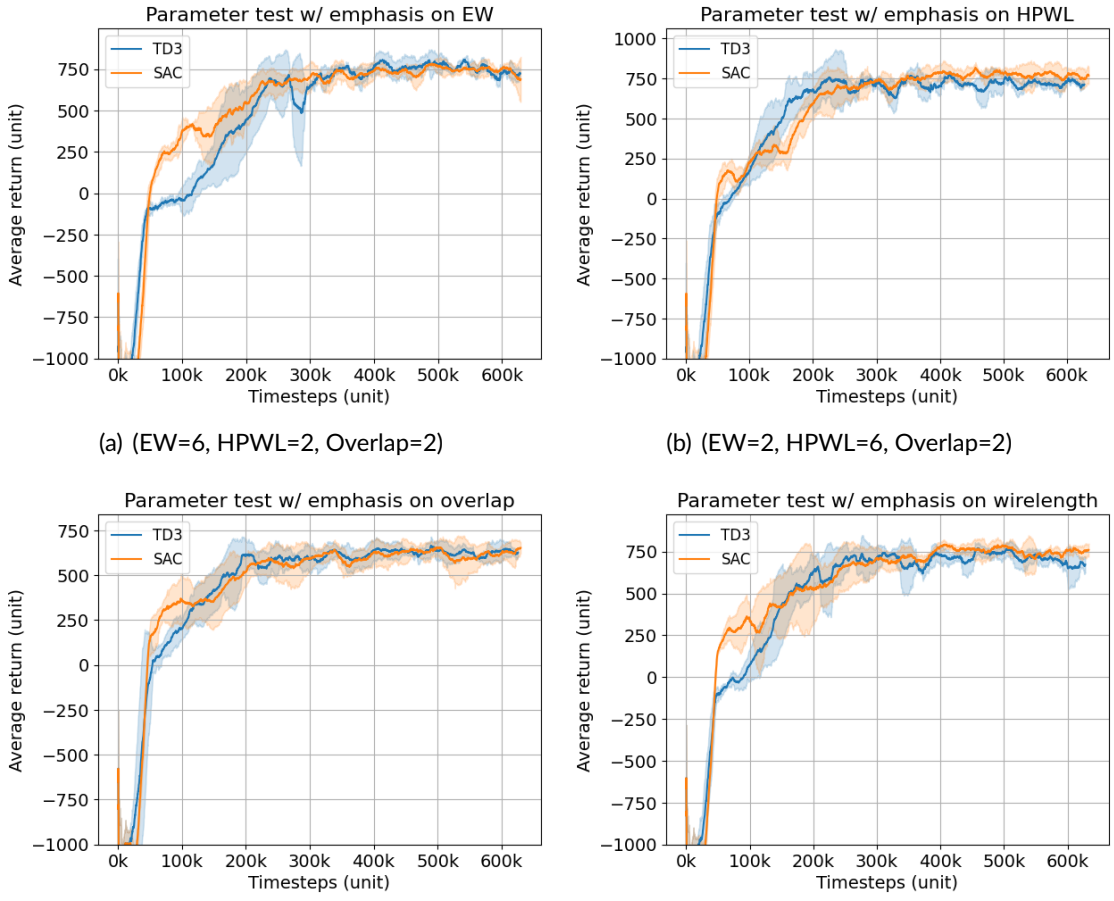


Figure 5.17: Experimental flow for iterative multi-component methodology. Experiments study the updated reward signal while supplementary experiments investigate particular features.

Multi-component experiments aiming to learn self-improving policies are presented in this section. This approach can overcome the limitations of the previously outlined single-component setup and generalise to unseen layouts. The outcomes introduced in this section may be categorised as follows: We start by presenting results and observing the learned behaviour as a consequence of emphasising specific parts of the reward function. Wirelength ablation tests investigating the characteristics of a reduced reward signal follow. The study also conducts supplementary experiments to compare the learning performance of an agent with pre-defined expert parameters against having to search and identify them as part of the interactive trial-and-error learning process. The size of the replay buffer, resizing strategy and their effect on learning are also studied. All evaluations presented are performed on unseen layouts and compared using post-routing wirelength. The flow chart in Figure 5.17 captures this flow pictorially.

5.3.1 Reward Function Parameter Trade-off Experiments

The reward signal consists of three tunable parameters, recalled from Equation 4.9a. The Euclidean Wirelength (EW) measures the wirelength to the component's nearest neighbours, the HPWL sums up the wirelength of all the nets the component is a part of, and the overlap quantifies any collision with nearby obstacles. This section presents the results for the four configurations described by Table 4.13, each favouring a particular parameter in the reward signal. The training performance is measured in terms of return, and each quoted result is an average derived from four distinct runs. The best policies are selected for further evaluation on unseen circuits, where their effectiveness is gauged using post-routing wirelength. SA-PCB (Holtz et al., 2020) is used to provide a baseline.



(a) (EW=6, HPWL=2, Overlap=2)

(b) (EW=2, HPWL=6, Overlap=2)

(c) (EW=2, HPWL=2, Overlap=6)

(d) (EW=4, HPWL=4, Overlap=2)

Figure 5.18: Average return for multi-component parameter experiments with TD3 and SAC. Averages are calculated from a maximum of four runs.

Configuration	TD3	TD3 (Pruned)	SAC	% SAC
EW=6, HPWL=2, Overlap=2	514.38 ± 359.81	739.09 ± 354.98	**750.20 ± 242.47**	1.48%
EW=2, HPWL=6, Overlap=2	525.74 ± 284.24	715.00 ± 302.43	**796.23 ± 270.45**	10.2%
EW=2, HPWL=2, Overlap=6	**623.46 ± 272.96**	**623.46 ± 272.96**	612.65 ± 262.53	-1.77%
EW=4, HPWL=4, Overlap=2	692.81 ± 348.48	692.81 ± 348.48	**743.65 ± 305.53**	6.84%

Table 5.13: Summary of average return for different multi-component parameter trade-off experiment configurations.

Figure 5.18 illustrates the average return after applying a moving average filter with a window of 100 samples for better visualisation. Runs that fail to learn a policy are removed from aggregation in the previous charts but retained in Table 5.13, summarising the average return. SAC leads TD3 on all configurations that emphasise wirelength and yield a 4.19% higher return on average.

The policy is evaluated against SA-PCB on unseen layouts M_u, and the performance is computed in terms of post-routing wirelength obtained after routing the circuit with PcbRouter (Lin et al., 2020). The results are summarised in Table 5.14. For experimental setups that heavily emphasise wirelength over overlap, the evaluation may fail to generate overlap-free results for all training runs. In such cases, the average is computed over the available runs. Policies that emphasise overlap tend to generate less optimised layouts regarding wirelength but are significantly more likely to yield overlap-free results while matching the results of SA-PCB. Overall, the second configuration (EW=2, HPWL=6, Overlap=2) trained with SAC outperforms SA-PCB on all unseen layouts by an average of 21.1% lower wirelength. Additionally, the remaining setups prioritising wirelength consistently outperform SA-PCB by a significant margin, regardless of the learning algorithm used. It should be noted that layout M_{U1} has only two components, and since SA-PCB performs a fixed amount of iterations, it will be relatively over-optimised.

	TD3			% TD3	SAC			% SAC
	M_{U0}	M_{U1}	M_{U2}		M_{U0}	M_{U1}	M_{U2}	
EW=6, HPWL=2, Overlap=2	30.8	**13.3**	54.8	15.9%	**30.9**	13.9	55.3	14.3%
EW=2, HPWL=6, Overlap=2	29.3	14.1	**50.4**	17.4%	34.5	**10.6**	51.7	21.1%
EW=2, HPWL=2, Overlap=6	36.7	16.2	63.8	-0.1%	40.9	15.6	62.3	-2.0%
EW=4, EW=4, Overlap=2	**27.9**	14.2	56.1	15.7%	36.4	12.9	**51.4**	13.9%
SA-PCB (Holtz et al., 2020)	38.9	**13.3**	75.3		38.9	13.3	75.3	

Table 5.14: Average post-routing wirelength from the best parameter trade-off experiment policies.

5.3.2 Ablation Experiments

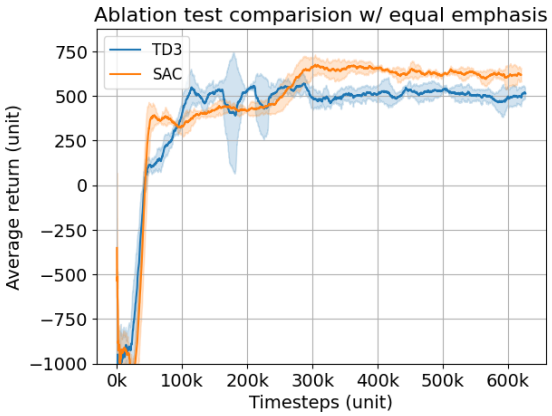
Ablation tests suppress a single component (EW or HPWL) in the reward signal to asses its contribution towards the performance objective. This section presents the experimental results for the ablation experiments described by Table 4.14. First, we assess the training performance regarding average return and then evaluate the best policies against SA-PCB (Holtz et al., 2020) on the unseen circuits in M_U ranked by post-routing wirelength. Lastly, we present a qualitative analysis demonstrating the placement optimisation capabilities of our policies. Using six cases, we illustrate the key moments, from a randomly initialised placement to an optimised layout observing both distinct fundamental behaviours embodied by policies and emergent collective strategies arising from the multi-component setup. All evaluations presented in this section are performed on unseen layouts.

Figure 5.19 illustrates the return charts plotted against the number of steps in the environment for the six ablation test configurations. Figures 5.19(a) to 5.19(c) omit the EW and respectively, equally weigh HPWL and overlap, emphasise HPWL and emphasise overlap. By contrast, Figures 5.19(d) to 5.19(f) omit HPWL and respectively, equally weigh the EW and overlap, emphasise EW and emphasise overlap. Table 5.15 summarises the return averaged over a maximum of four runs. TD3 outperforms SAC by 3.5% on average for configurations that linearly combine EW and overlap, while SAC lead by an average of 6.1% on the rest. Additionally, SAC exhibits less variation during training and is less sensitive to initial random conditions, evident from the lower standard deviation.

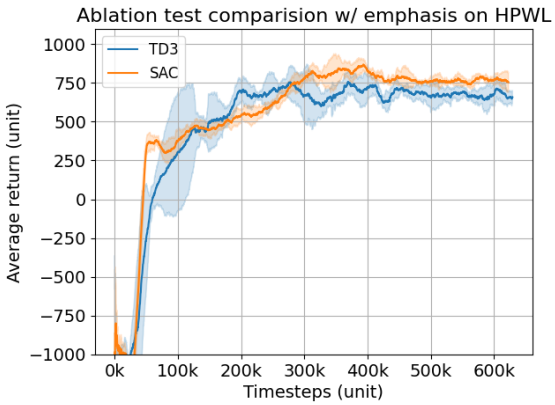
Routed wirelength values attained by the best policies in each category are presented in Table 5.16. Setups that emphasise wirelength have better performance, albeit less likely to generate overlap-free layouts. Overall, SA-PCB outperforms policies that heavily emphasise overlap. By contrast, policies trained by TD3 and SAC emphasising HPWL on average outperform SA-PCB by 16% and 15%, respectively. Those that emphasise EW obtain a slightly lower margin of 14.3% and 11% for TD3 and SAC. Configurations that give equal importance to wirelength and overlap outperform the baseline by 2.7%.

Configuration	**TD3**	**TD3 (Pruned)**	**SAC**	**% SAC**
HPWL=5, Overlap=5	510.22 $\pm$ 283.69	510.22 $\pm$ 283.69	**578.16 $\pm$ 232.07**	11.75%
HPWL=2, Overlap=8	498.65 $\pm$ 311.67	**825.32 $\pm$ 257.57**	780.23 $\pm$ 246.30	-5.78%
HPWL=8, Overlap=2	305.34 $\pm$ 321.68	632.36 $\pm$ 366.32	**720.11 $\pm$ 331.52**	12.19%
EW=5, Overlap=5	**729.98 $\pm$ 326.45**	**729.98 $\pm$ 326.45**	712.74 $\pm$ 367.55	-2.42%
EW=2, Overlap=8	785.78 $\pm$ 322.92	**1007.95 $\pm$ 294.59**	942.54 $\pm$ 248.74	-6.94%
EW=8, Overlap=2	**812.94 $\pm$ 366.38**	**812.94 $\pm$ 366.38**	802.84 $\pm$ 356.93	-1.26%

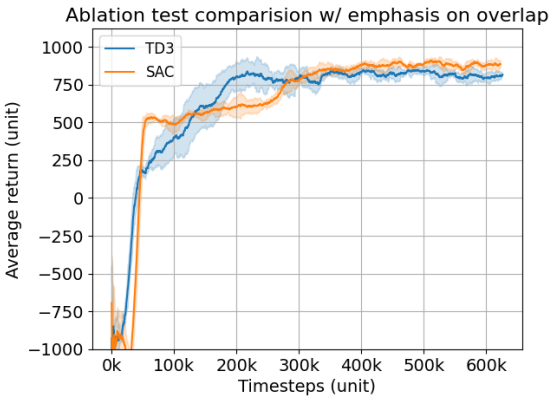
Table 5.15: Summary of average return for multi-component ablation experiments.



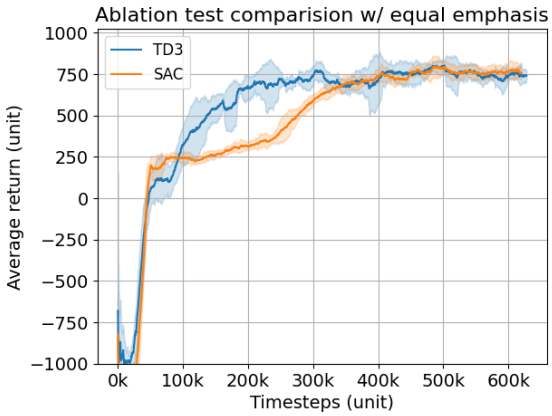
(a) (EW=0, HPWL=5, Overlap=5)



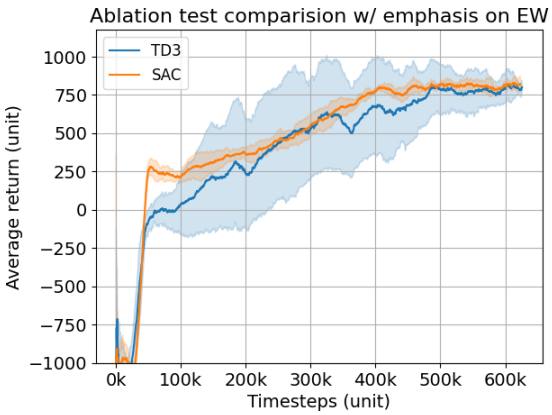
(b) (EW=0, HPWL=8, Overlap=2)



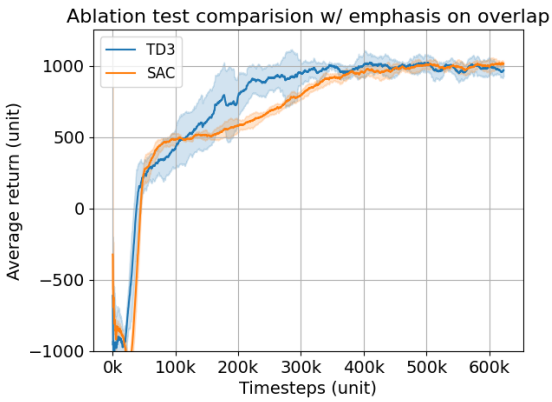
(c) (EW=0, HPWL=2, Overlap=8)



(d) (EW=5, HPWL=0, Overlap=5)



(e) (EW=8, HPWL=0, Overlap=2)



(f) (EW=2, HPWL=0, Overlap=8)

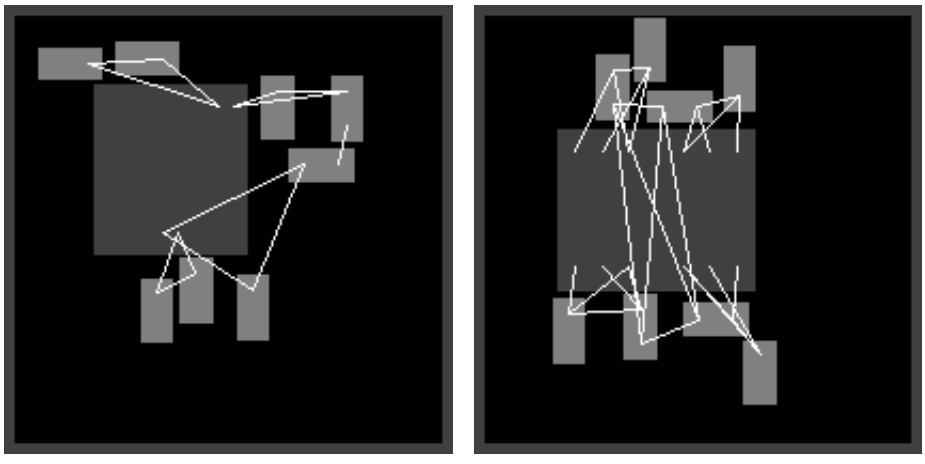
Figure 5.19: Average return for multi-component parameter experiments with TD3 and SAC. Average calculated over a maximum of four runs.

	TD3			% TD3	SAC			% SAC
	M_{U0}	M_{U1}	M_{U2}		M_{U0}	M_{U1}	M_{U2}	
HPWL=5, Overlap=5	45.0	15.9	61.8	-5.6%	38.1	15.6	59.6	2.0%
HPWL=2, Overlap=8	44.9	15.6	86.0	-15.4%	59.6	16.3	73.8	-24.5%
HPWL=8, Overlap=2	**29.0**	15.5	46.0	16.0%	**28.7**	15.2	**50.8**	15.0%
EW=5, Overlap=5	35.5	16.9	64.4	-1.3%	36.1	15.5	61.3	3.3%
EW=2, Overlap=8	43.2	15.4	71.5	-7.1%	48.3	16.4	71.7	-14.0%
EW=8, Overlap=2	32.7	15.5	**42.7**	14.3%	29.3	15.1	58.4	11.3%
SA-PCB (Holtz et al., 2020)	38.9	**13.3**	75.3		38.9	**13.3**	75.3	

Table 5.16: Average post-routing wirelength generated by the best policies from the ablation experiments.

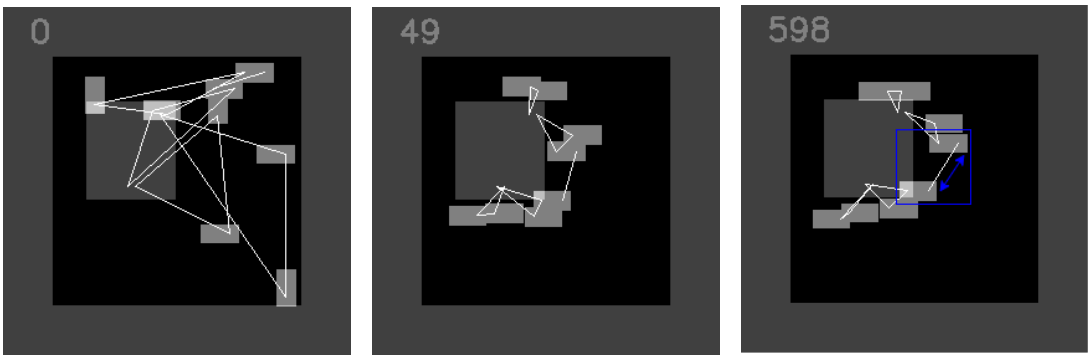
5.3.3 Qualitative Policy Analysis

A quantitative analysis of particular policies on unseen circuits M_{U0} and M_{U2} is presented in this section. M_{U1} is omitted because it presents a trivial case. We will analyse fundamental behaviours exhibited by the policies that suggest they learned the fundamental dynamics of the task. Additionally, we will highlight emergent collaborative and competitive behaviour conditionally evoked by emphasising HPWL and EW, respectively. In all scenarios, we start from a random initial placement and, over 600 steps, progress towards the terminal condition, where a stable, optimised layout should be reached. Figure 5.20 provides the unseen circuits optimised by SA-PCB over 600 iterations after starting from an identical initial state. They are provided as a reference and to highlight the benefit of



(a) M_{U0} - Step 600/600 (SA-PCB) (b) M_{U2} - Step 600/600 (SA-PCB)

Figure 5.20: Circuit placements optimised with SA-PCB over 600 steps.

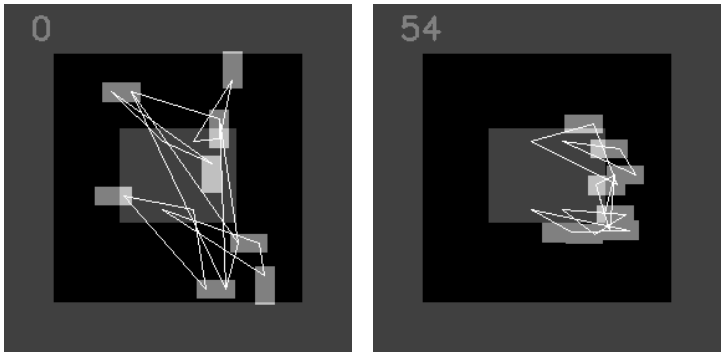


(a) Initialisation - Step 0/600 (b) Step 49/600 (c) Step 598/600

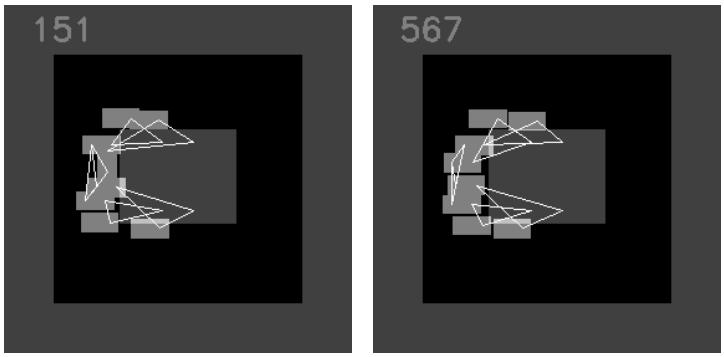
Figure 5.21: Key states in optimising M_{U0} with policy (EW=8, Overlap=2).

leveraging experience over the stochastic nature of meta-heuristics.

Figures 5.21 and 5.22 analyse a policy that greedily optimises EW over overlap. Regarding the former, the system quickly settles to a suitable placement, starting from a ran-



(a) Initialisation - Step 0/600 (b) Step 54/600

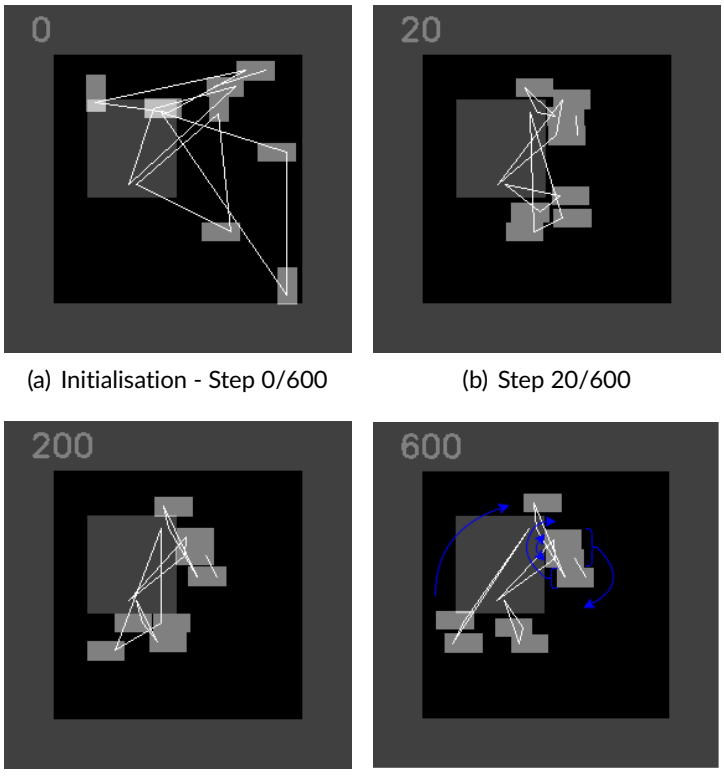


(c) Step 151/600 (d) Step 567/600

Figure 5.22: Key states in optimising M_{U2} with policy (EW=8, Overlap=2).

dom initial placement. The policy was rewarded when taking actions that yielded reduced wirelength solely for the constituent component and without regard to its neighbours. Regardless of the greedy strategy, dependent component clusters emerge. Interestingly, the conflicting situation in which the components enclosed in the blue box of Figure 5.21(c) is resolved differently than that in Figure 5.25(f). Later we will show policies that assign higher importance to HPWL and will cooperate in such a scenario. However, in this scenario, we noticed competition. The components oscillate back and forth in a push-pull manner without resolving to a stable positioning.

The policy in emphasising EW in Figure 5.22 is evaluated on layout M_{U2}, which contains many small component clusters. Starting from a randomised placement in Figure 5.22(a), an initial placement emerges quickly after only 54 steps. However, this was not good enough, and all the components were driven towards the opposite side moving from 5.22(b) to 5.22(c). While the position in 5.22(d) is better than 5.22(b), the latter is a rotation away from achieving a similar result. The surprisingly good performance of this policy on a layout with many multi-pin nets may be attributed to its chaotic behaviour,



(a) Initialisation - Step 0/600 (b) Step 20/600

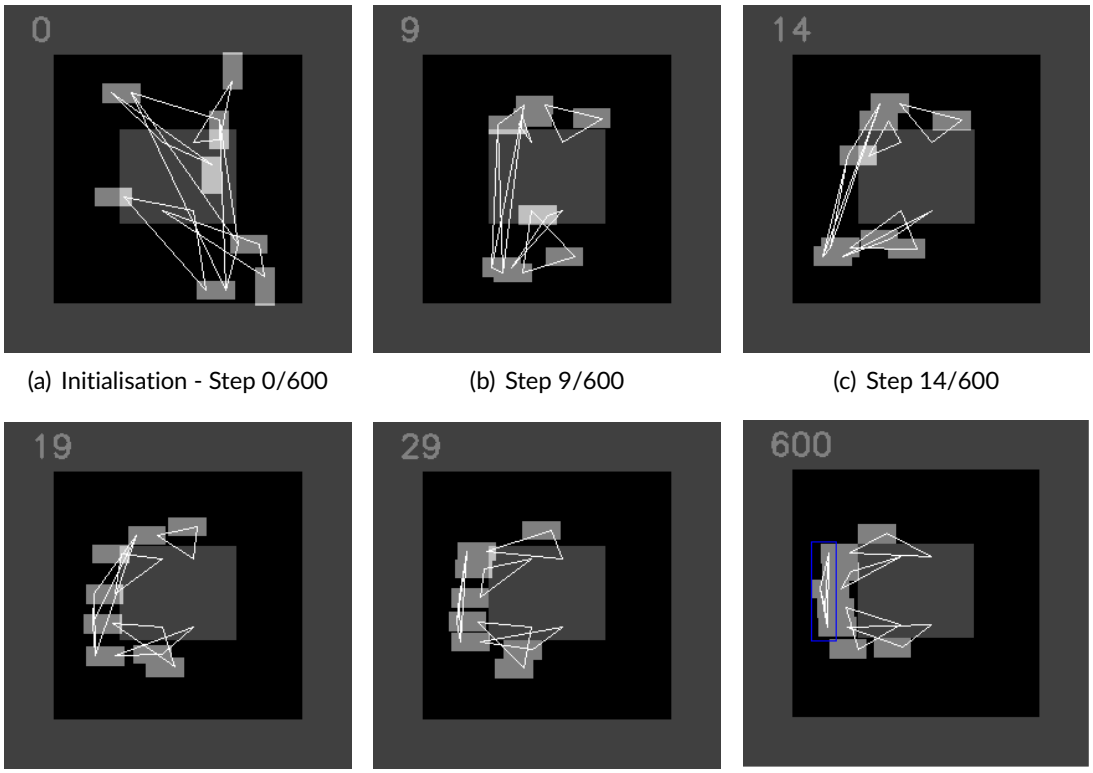
(c) Step 200/600 (d) Final state - Step 600/600

Figure 5.23: Key states in placing M_{U0} with policy (HPWL=5, Overlap=5).

which encourages exploration. It emerges from the fact that each component carries out the best action for itself without regard to its neighbours.

Figure 5.23(a) illustrates a randomly initialised placement from which two clusters quickly emerge, although the resulting choice of constituents could be better. It takes the policy close to 200 steps for a minor improvement moving from Figure 5.23(b) to 5.23(c). Similar to the change from 5.23(c) to 5.23(d). The blue arrows in Figure 5.23(d) highlight a possible strategy that may lead to a better placement. The movement requires either circling the neighbouring components or moving through them. The former requires temporarily acquiring a penalty due to worsening the wirelength, and similarly for the latter, albeit due to overlap. Since the reward signal balances the two, the policy will be severely penalised for performing either and thus is not driven to pursue these potential scenarios.

Figure 5.24 captures the seminal steps for the same policy on layout M_{U2}. In the initial steps, the policy quickly divides into two clusters as shown by 5.24(b) and 5.24(c) such that a few steps later, the resulting placement is a near-perfect one as shown by 5.24(d). In Figure 5.24(e), after only 29 steps, the optimised layout generated by our policy resembles



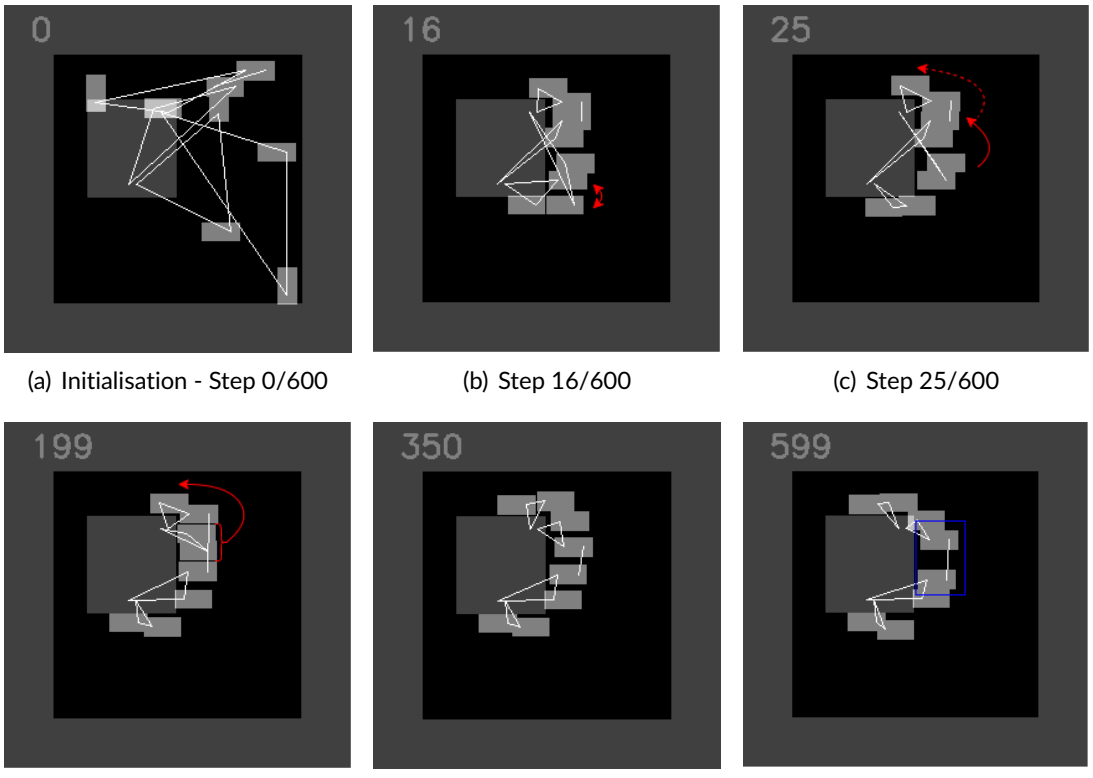
(a) Initialisation - Step 0/600 (b) Step 9/600 (c) Step 14/600

(d) Step 19/600 (e) Step 29/600 (f) Step 600/600

Figure 5.24: Key states in placing M_{U2} with policy (HPWL=5, Overlap=5).

the final one 5.24(f). By contrast, the one generated by SA was not as optimised after 600 steps, a 20x difference. Notice how components connected by the net enclosed within the blue rectangle come close together in Figure 5.24(f). Until the episode terminates, minor changes are observed.

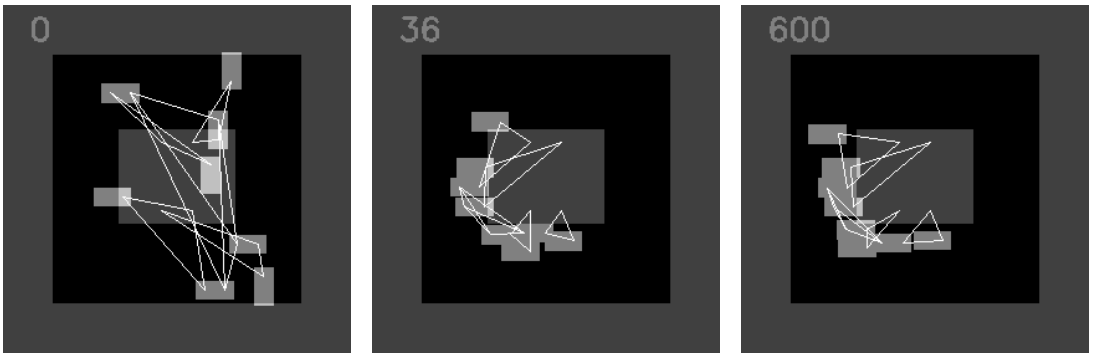
Next, Figures 5.25 and 5.26 demonstrate a policy emphasising wirelength. Placement starts randomly in Figure 5.25(a). Notice identical initialisation conditions to Figure 5.24(a). The components quickly move towards the anchoring IC in 5.25(b). Since this is a poor placement, a set of swaps marked by red arrows are witnessed in 5.25(c) and 5.25(d). These are motivated by a decrease in HPWL and are evident in 5.25(e) by component clusters closely collating around the anchor point. A conflicting situation emerges towards the end of the episode, marked by the blue square in Figure 5.25(f). The members seem to take a side and stick to it, a collaborative effort - this is the behaviour observed in the video these frames were extracted from. Furthermore, it can be seen that the layout degrades moving from step 350 to 600 with some overlap, which is tolerated due to less importance being associated with it in the reward function.



(a) Initialisation - Step 0/600 (b) Step 16/600 (c) Step 25/600

(d) Step 199/600 (e) Step 350/600 (f) Step 599/600

Figure 5.25: Key states in placing M_{U0} with policy (HPWL=8, Overlap=2).



(a) Initialisation - Step 0/600 (b) Step 36/600 (c) Final state - Step 600/600

Figure 5.26: Key states in optimising circuit M_{U2} with a HPWL greedy policy (HPWL=8, Overlap=2).

Figure 5.26(a) quickly converges to a good layout. However, the previous policy performs better on the same layout. The larger triangles representing a net are noticeably more significant, leading to a higher wirelength approximation. Furthermore, up to 10% overlap is present in the accompanying quoted HPWL, whereas none was present prior.

5.3.4 Key Conclusions for Multi-Component Iterative Placement

We demonstrated a multi-component environment derived from strong baselines developed in the previous section, along with implementations of TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018). The infrastructure was used with a revised reward signal (Equation 4.10) derived from R_3 to train general policies. Parameter tests emphasised particular aspects of the reward function, and ablation tests studied the placement behaviour in constrained contexts. Supplementary experiments studied the impact of starting with expert knowledge and the effect of the replay buffer size and resizing strategy.

The variety of data points arising from different circuits under the guidance of an adaptive and unified reward signal enabled the policy to learn generalisable placement techniques. The resulting layouts surpassed SA-PCB (Holtz et al., 2020) by 17% and 21% for policies trained using TD3 and SAC, respectively, measured with routed wirelength. All evaluations were performed on layouts not seen during training. Training processes governed by SAC were notably more stable and tended to outperform TD3 in average return. Furthermore, when analysing the policy patterns in Figures 5.23 - 5.22, elements of cooperation and competitiveness could be identified as consistent with the emphasis on specific aspects of the reward signal. Notable policies were observed to undergo sequences of actions that yielded low instantaneous rewards, eventually leading to bet-

ter placements. This analysis was accompanied by an extensive set of experiments that meet the requirements set by objective four to design and evaluate a multi-component RL setup for iterative PCB component placement capable of generalisable behaviour.

5.4 Summary

The constructive placement methodology suggested by Mirhoseini et al. (2021) lacks key information concerning its methodology and mainly requires improving its weak evaluation (Cheng et al., 2023; Xu et al., 2022). In our attempts to apply this approach, we trained a circuit quality predictor achieving respective accuracies of 72% and 69% for HPWL and post-routing wirelength on unseen circuits. While better features (Xie et al., 2021; Zhang et al., 2020) can improve generalisation performance, this was used to encode the observation space of our placer. We concluded that this methodology has significant scalability challenges for PCB layout, and our simplified approach did not provide sufficient evidence of learning placement behaviour using quantitative and qualitative assessments.

Single-component iterative placement demonstrated learning policies in constrained PCB setups which validated our MDP formulation. We studied the efficacy of learning policies by mimicking human experts and concluded that while it was successful in particular instances, generalisation was not feasible due to inconsistent design styles exhibited by human experts. As a result, the policy receives mixed signals from the reward signal and observation space (the fixed portion of the layout). We partly addressed the issue through an adaptive reward signal and demonstrated that distinct placement behaviours could be evoked within the constrained problem setup. Despite some limitations, the constrained problem setup provided a perfect environment to study core parameters and their interactions. The results influenced the decisions that set up the foundations of the main contribution of this thesis: multi-component iterative placement methodology.

We achieved consistent learning yielding general policies after addressing the remaining inconsistencies through a multi-component setup, where all netlist components contribute to learning a policy. This adds the benefit of generating highly diverse data, as each component brings unique perspectives aligned with the current policy. Quantitative tests demonstrated a 17% and 21% improvement for TD3 and SAC respectively over SA-PCB (Holtz et al., 2020) in terms of post-routing wirelength. Qualitative assessments demonstrated that fundamental placement techniques were learned and revealed emergent properties of collaboration and competition. They also suggested faster placement convergence that sometimes exceeded an order of magnitude over SA-PCB.

6 Conclusions

This thesis aimed to identify novel methods for PCB component placement with RL capable of learning generalisable placement techniques. In the following sections, we discuss the contributions, limitations and present avenues for further research.

6.1 Revisiting the Aims and Objectives

In this section, we revisit the objectives of this thesis and describe our approach to achieving them. Objective four is the culmination and main contribution where we trained RL policies that learned general behaviour for PCB component placement. To our knowledge, this work is the first in the EDA community to use RL for an end-to-end AI workflow.

6.1.1 Constructive Placer

Inspired by the work of Mirhoseini et al. (2021), the first objective aimed to assess the feasibility of an RL based constructive component placer by using a circuit quality estimator as the encoder of the policy. We adapted the current state-of-the-art AI-assisted workflow, initially targeting the floorplanning step in the IC design flow (Mirhoseini et al., 2021) for PCB component placement. A GNN was trained to predict wirelength, achieving an accuracy of 88.37%, and was used to measure placement quality. With the prediction layer removed, it was integrated into a custom RL environment with our C++ placement engine and encoded the problem state. The best-trained policies with TRPO showed little evidence of learning and were able to perform component placement, albeit falling behind SA-PCB (Holtz et al., 2020), a SA placer, by 35% when compared using HPWL. This objective led to the development of the following tools:

1. OpenAI Gym (Brockman et al., 2016) constructive PCB placement environment incorporating a C++ placement engine and placement quality estimator.

2. Dataset of 30 real-world circuits in KiCad (Bautista et al., 2022) format.

3. Configurable dataset generation tool, that arbitrary generates partial placements and optimises them with SA-PCB (Holtz et al., 2020) to varying degrees.

4. Improved the KicadParser (Yenyi et al., 2020) library, SA-PCB (Holtz et al., 2020) SA-based PCB placer and A* PcbRouter (Lin et al., 2020). Improvements described in Appendix A.2.

5. Created a high-performance representation system for circuit boards in C++. This was packaged in a way that facilitated working with many layouts of arbitrary size. These software libraries are version controlled, documented with Doxygen and integrated into Python through SWIG (Beazley, 1996).

6.1.2 Formulating the PCB Component Problem as an RL task

The iterative PCB placement problem was formulated as an RL task. The evaluation was primarily conducted in Section 5.2.1 and applied to single-component setups in Section 5.2. It was then adapted for multi-component setups in Section 5.3.

6.1.3 Design and Testing of a Single-Component PCB Placer

The single-component placement methodology saw the development of an RL environment and systematically studied its parameters, three fundamentally different reward signal categories, discrete and continuous action spaces and four learning algorithms. Experiments motivating agents to mimic expert designers suggested that on-policy algorithms TRPO and PPO were only successful with discrete action spaces. At the same time, hybrid actor-critic methods yielded the best performance with continuous action spaces. While both methodologies yielded good performance on individual layouts, generalisation performance was poor. Identifying the cause as inconsistencies arising from expert data leaking into the problem through the reward signal and observation space (the fixed portion of the layout), R_3 addressed the former by assigning credit based on the agent's expertise. Final results demonstrated that desirable circuit placement behaviours could be evoked, albeit again falling short on generalisation performance. The granular control offered by continuous action space led to superior performance when combined with TD3 and SAC, which were retained for future experiments due to their distinctive properties. This objective necessitated the development of the following tools:

1. Configurable OpenAI Gym (Brockman et al., 2016) environment for iterative single-component placement.

2. Fully automated, highly parallelisable experimental and testing procedures with automated result aggregation and `.pdf` report generation.

6.1.4 Multi-Component RL Capable of Generalised PCB Placement

The multi-component environment was adapted from the last objective for addressing the data inconsistencies from the fixed portion of the layout that leaked through the observation space. We have engineered a reward signal (Equation 4.10) that allowed training across distinct circuit layouts and learning generalisable placement techniques. Experiments displayed that general policies could be learned and emergent collaborative or competitive behaviour could be evoked conditional on the emphasised parameters in the reward signal. Results on average showed 17% and 21% (Table 5.16) reduction in post-routing wirelength compared with SA-PCB for policies trained with TD3 and SAC, respectively. Faster convergence to solution was observed to be more than an order of magnitude. We attribute these improvements to selecting RL as a method of optimisation since the strategy of measuring task performance is similar in both cases. Our approach towards generalisation required the agents to autonomously identify what makes a good placement and then learn placement policies. Additionally, the placement scenarios collected must be consistent with the agent's definition of 'good' placement. We achieved this by removing all expert-attributed elements and allowing the policy to control all components. The attainment of this objective is the main contribution of this work, and is the first to demonstrate the use of RL for learning general policies that automate placement tasks end-to-end. This objective delivered the following:

1. A custom multi-component environment for iterative PCB component placement
2. Vanilla implementation of TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) with a resizeable replay buffer and associated resizing strategies.
3. Fully automated, highly parallelisable experimental and testing procedures with automated result aggregation and `.pdf` report generation.

6.2 Limitations

This thesis brings relevant contributions to the field of EDA but is not free from limitations. In this section, we outline the shortcomings of the studied methodologies and suggest potential solutions to overcome them.

6.2.1 Limitations of Constructive Placement

We witnessed poor layout performance with no clear evidence of generalisation in our approach derived from the state-of-the-art AI floorplanner (Mirhoseini et al., 2021). This research has drawn much praise (Hao et al., 2022; Huang et al., 2021; Khailany et al., 2020; Lopera et al., 2021) for being innovative and the first to automate an EDA task in an end-to-end using AI techniques. However, it was also critiqued for its inability to generalise (Crocker, 2021; Xu et al., 2022). We have found that the research makes ambitious claims that were not reflected in the adopted processes. For instance, the results were reported after a process coined domain adaptation in which a pre-trained policy was fine-tuned on the circuit that would later be used for its evaluation. Additionally, while the general methodology was presented, the details were not adequately documented, making it difficult for us to replicate their work without assumptions. Cheng et al. (2023) assessed the methodology and concluded that their results were not reproducible on open benchmarks. Despite this, limitations to scale to practical problem sizes were discussed in detail in Section 5.1.3, including the conflict between an extensive action space and resolution. This thesis has shown that iterative placement is superior to constructive approaches on small circuits, and while the latter has its merits, perhaps it is better utilised in a hybrid approach or as an alternative to random initialisation.

6.2.2 Evaluation in terms of Post-Routing Wirelength

While we averaged multiple placement runs starting from different initial conditions and configured the router to make extensive efforts in finding optimal routes, the strategy has potential for improvement. Future evaluation strategies should include different routing tools, for instance, the open-source Freerouting (Wirtz, 2023) and emerging commercial alternatives , to increase confidence in the results and minimise potential biases. Such a strategy has also been witnessed in recent literature by Cheng et al. (2022).

6.2.3 Sub-Optimal Weighting of Adaptive Reward Parameters

We observed that the best performance was obtained by associating a high cost with wirelength at the expense of overlap. The result yielded placements with shorter wirelength than SA, albeit with a lower frequency of overlap-free layouts. Prioritising the wirelength dimensions over the overlap term in the multi-objective function is advantageous because future work can implement a post-processing legalisation step to resolve minor overlaps. However, since two flavours of wirelength are introduced the trade-off between the two is underexplored and is potentially dependent on the particular con-

nectivity of a given component. Literature tackling similar problems (Ismail et al., 2012) showed that wrapping the weighting of individual parameters in the cost functions into yet another optimisation process may yield improved performance. Our approach of using pre-defined weighting places the performance evaluation at a disadvantage since more sophisticated approaches like Pareto-based optimization methods or evolutionary algorithms may lead to better results regarding wirelength performance and consistency of overlap-free placements.

6.3 Future Work

We conclude this thesis by briefly discussing several potential avenues for future research. The modular and automated design methodology in Chapter 4 offers ample flexibility for integrating improvements. The independent evaluations for each task in Chapter 5 can help guide our recommendations for where future effort should be invested.

6.3.1 Expand the Multi-Component Setup

The multi-component setup was constrained in the interest of time and to limit our scope by focusing on generalisation to unseen layouts. Two improvements can be made to the policy to enhance its applicability to a larger class of circuits. First, the policy's action space currently restricts placement to a single side of the PCB and would require the addition of an extra output that dictates on which side the component is placed. Secondly, the observation space only supports two-terminal components and can be expanded to work with multi-pin devices to cater for more circuits. Including these two features will create a much larger class of circuits with even more significant practical applicability.

Our application to general-purpose component placement represents a proof of concept with implications for more ambitious co-optimisation applications, including thermal-aware placement, high power placement and high-speed layout. Limited time and computation resources restricted further improvements to problem setup and more extensive hyperparameter optimisation. Furthermore, parallelising the data collection process will decrease training runtime and expedite iterative prototyping cycles. Consequently, our research presents numerous opportunities for additional investigation, with significant practical applicability and potential for commercialisation.

Minimise the gap between an academic RL task and practical applicability by robustly integrating the policy as a KiCad (Bautista et al., 2022) plugin. While our result is a proof of concept and may require further development before this step is feasible, engaging with the public will ensure that the research remains focused on solving real-world circuits and

is met with timely and appropriate critique from the community. Additionally, the open-source community may alleviate some of the challenges associated with the long-term accessibility and maintainability of software tools.

6.3.2 Investigate the Offline Reinforcement Learning

Our single-component experiments which assigned credit depending on how well the agent's placement reflected that of the expert designer, suggested significant limitations to generalisation due to inconsistencies arising from different expert designers (Section 5.2.6). However, key trajectories from the multi-component learning setup can be collected and through offline RL used to learn potentially better general placement policies while also reducing learning time. First, we have outlined that it can be challenging to learn good policies, which is exacerbated by long training time due to image-processing-based methods for deriving observations. Secondly, our demonstrations illustrated that collaborative or competitive placement styles could be evoked thus a potentially better general policy can be learned from a dataset crafted from many training runs. Such an approach can potentially yield better general policies and online learning can be reserved for fine-tuning policies for particular domains. For instance, placement for digital, analogue or mixed-signal circuits, which have unique characteristics and also, placement co-optimisation with thermals or parasitics amongst many others.

6.3.3 Improve Feature Extraction and Move Beyond Wirelength

We provide three pieces of information to the RL agent. A description of the environment surrounding the component (Section 4.2.3.1 and 4.3.2), vector information for guidance to the goal area (Section 4.2.3.2) and information relating to its size and positioning and orientation. These features are manually derived, and literature shows that an end-to-end learning approach may yield better results (Badia et al., 2020; Mnih et al., 2013, 2015). More specifically, CNNs may be used to extract translation and rotation invariant features from a limited visual field surrounding the component. GNNs may be used to extract contextual positional information while considering all pins connected by a net.

Convolution methods on graph data have been successful in a wide range of prediction tasks, including net length and critical path estimation (Xie et al., 2021), routability (Kirby et al., 2019) and power estimation (Zhang et al., 2020), amongst others. GNNs may also be useful for understanding relations between components and, through RL, make decisions that are not solely motivated by wirelength minimisation. PCB design engineers rarely think in terms of wirelength except for high-speed layout. This information may be used

to motivate novel placement behaviours that are not driven by wirelength but by problem constraints. For instance, a decoupling capacitor should introduce as little wirelength as possible for physical reasons (a small inductance is associated with a short trace and will allow the delivery of power bursts with minimal resistance to current). However, by design, a low-speed interface connector may be required at a specific location on the circuit board that introduces significant wirelength.

6.3.4 Improve Policy Performance

The seminal Neuroevolution (Stanley and Miikkulainen, 2002) generated exciting results on small problems. However, even with an indirect encoding scheme (Stanley, 2007), it failed to scale to more extensive networks necessary for problems with large state spaces such as the one tackled in this thesis. Evolutionary Reinforcement Learning (ERL) is an emerging area of research that takes a hybrid approach by combining gradient-based and evolutionary learning. Sigaud (2022) survey a broad body of literature on the topic and classify algorithms into four categories. The notable work by Bodnar et al. (2020) uses DDPG (Lillicrap et al., 2015) in conjunction with an improved GA that is adapted for reproducing neural networks without the catastrophic forgetting inherent to mutation and crossover operations. As a result, they propose more sophisticated genetic operators that allow sufficiently large policies to train and scale better. The original authors note that their novel ERL outperforms PPO (Schulman et al., 2017) and TD3 (Fujimoto et al., 2018) on all five robot locomotion environments from the OpenAI Gym Brockman et al. (2016). Our research has outlined the importance of varied data points that consistently represent the problem. Using a more rigorous population-based learning method is another way to promote learning distinct behaviours from the same neural network parameters.

More extensive training can also be performed by introducing randomness to overcome the effects of initial conditions. Components in the multi-component approach were observed undergoing a sequence of deteriorating actions in pursuit of a placement that better co-locates them to their neighbours (Figure 5.25). This was especially evident in scenarios where wirelength was emphasised. However, as noted earlier, it tends to make it harder to achieve overlap-free layouts. Therefore adding more variation during training through perturbation (e.g. randomly swapping two components) may provide examples of challenging situations where the policy may learn increasingly robust placement techniques. During inference, we expect this to motivate the policy to make a sequence of actions that will ultimately lead to consistently better results.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Adya, S. N. and Markov, I. L. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of the 2002 International Symposium on Physical Design*, ISPD '02, page 12–17, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134606. doi: 10.1145/505388.505392.

Adya, S. N., Chaturvedi, S., Roy, J. A., Papa, D. A., and Markov, I. L. Unification of partitioning, placement and floorplanning. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '04, page 550–557, USA, 2004. IEEE Computer Society. ISBN 0780387023. doi: 10.1109/ICCAD.2004.1382639.

Agnesina, A., Chang, K., and Lim, S. K. Vlsi placement parameter optimization using deep reinforcement learning. pages 1–9. ACM, 11 2020. ISBN 9781450380263. doi: 10.1145/3400302.3415690.

Agnesina, A., Rajvanshi, P., Yang, T., Pradipta, G., Jiao, A., Keller, B., Khailany, B., and Ren, H. Autodmp: Automated dreamplace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, ISPD '23, page 149–157, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399784. doi: 10.1145/3569052.3578923.

Agnihotri, A., Yildiz, M., Khatkhate, A., Mathur, A., Ono, S., and Madden, P. Fractional cut: improved recursive bisection placement. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, pages 307–310, San Jose, CA, USA, 2003. IEEE. ISBN 978-1-58113-762-0. doi: 10.1109/ICCAD.2003.1257685.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701.

Alexandridis, A., Paizis, E., Chondrodima, E., and Stogiannos, M. A particle swarm optimization approach in printed circuit board thermal design. *Integrated Computer-Aided Engineering*, 24:143–155, 3 2017. ISSN 10692509. doi: 10.3233/ICA-160536.

Alpert, C. J. The ispd98 circuit benchmark suite. In *Proceedings of the 1998 International Symposium on Physical Design*, ISPD '98, page 80–85, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 158113021X. doi: 10.1145/274535.274546.

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2001–2009, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Badriyah, T., Setyorini, F., and Yuliawan, N. The implementation of genetic algorithm and routing lee for pcb design optimization. *2016 International Conference on Informatics and Computing, ICIC 2016*, pages 148–153, 2017. doi: 10.1109/IAC.2016.7905706.

Bautista, R. F., Charras, J.-P., Evans, J., Hillbrand, S., McInerney, I., Pointhuber, T., Roszko, M., Stambaugh, W., Wielgus, M., Wlostowski, T., and et al. Kicad eda, Nov 2022.

Beazley, D. M. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, page 15, USA, 1996. USENIX Association.

Bergstra, J., Yamins, D., and Cox, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013. doi: 10.5555/3042817.3042832.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993. doi: 10.5555/2986459.2986743.

Bishop, C. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, 2016. ISBN 9781493938438.

Bodnar, C., Day, B., and Lió, P. Proximal distilled evolutionary reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3283–3290, 4 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i04.5728.

Bradski, G. and Kaehler, A. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

Brenner, U. and Struzyna, M. Faster and better global placement by a new transportation algorithm. In *Proceedings of the 42nd Annual Design Automation Conference*, DAC '05, page 591–596, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930582. doi: 10.1145/1065579.1065733.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34:18–42, 7 2017. ISSN 1053-5888. doi: 10.1109/MSP.2017.2693418.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 12 2013. doi: 10.48550/arxiv.1312.6203.

Bustany, I. S., Chinnery, D., Shinnerl, J. R., and Yutsis, V. Ispd 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, page 157–164, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333993. doi: 10.1145/2717764.2723572.

Chan, T., Cong, J., and Sze, K. Multilevel generalized force-directed method for circuit placement. In *Proceedings of the 2005 International Symposium on Physical Design*, ISPD '05, page 185–192, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930213. doi: 10.1145/1055137.1055177.

Chan, T. F., Sze, K., Shinnerl, J. R., and Xie, M. *mPL6: Enhanced Multilevel Mixed-Size Placement with Congestion Control*, pages 247–288. Springer US, 2007. ISBN 978-0-387-68739-1. doi: 10.1007/978-0-387-68739-1_10.

Chan, W.-T. J., Ho, P.-H., Kahng, A. B., and Saxena, P. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, ISPD '17, page 15–21, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346962. doi: 10.1145/3036669.3036681.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785.

Chen, T. C., Jiang, Z. W., Hsu, T. C., Chen, H. C., and Chang, Y. W. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27:1228–1240, 2008. ISSN 02780070. doi: 10.1109/TCAD.2008.923063.

Cheng, C. K., Kahng, A. B., Kang, I., and Wang, L. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38:1717–1730, 2019. ISSN 19374151. doi: 10.1109/TCAD.2018.2859220.

Cheng, C. K., Ho, C. T., and Holtz, C. Net separation-oriented printed circuit board placement via margin maximization. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2022-Janua:288–293, 2022. doi: 10.1109/ASP-DAC52403.2022.9712480.

Cheng, C.-K., Kahng, A. B., Kundu, S., Wang, Y., and Wang, Z. Assessment of Reinforcement Learning for Macro Placement. March 2023. arXiv:2302.11014 [cs].

Cheng, H. C., Huang, Y. C., and Chen, W. H. A force-directed-based optimization scheme for thermal placement design of mcms. *IEEE Transactions on Advanced Packaging*, 30:56–67, 2007. ISSN 15213323. doi: 10.1109/TADVP.2006.890211.

Chiou, C. H., Chang, C. H., Chen, S. T., and Chang, Y. W. Circular-contour-based obstacle-aware macro placement. volume 25-28-January-2016, pages 172–177. Institute of Electrical and Electronics Engineers Inc., 3 2016. ISBN 9781467395694. doi: 10.1109/ASPDAC.2016.7428007.

Cohoon, J. P., Hegde, S. U., Martin, W. N., and Richards, D. S. Distributed genetic algorithms for the floorplan design problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10:483–492, 1991. ISSN 19374151. doi: 10.1109/43.75631.

Cohoon, J. and Paris, W. Genetic placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6:956–964, 11 1987. ISSN 0278-0070. doi: 10.1109/TCAD.1987.1270337.

Crocker, P. *Physically Constrained PCB Placement Using Deep Reinforcement Learning*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2021.

Davidson, S. Itc'99 benchmark circuits - preliminary results. In *International Test Conference 1999. Proceedings (IEEE Cat. No.99CH37034)*, pages 1125–1125, 1999. doi: 10.1109/TEST.1999.805857.

DeepPCB. Pure ai powered, cloud-naitve pcb routing. `https://deeppcb.ai`, 2023.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*. arXiv, 2019. doi: 10.48550/arxiv.1903.02428.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.

Gao, X., Zhang, H., Liu, M., Shen, L., Pan, D. Z., Lin, Y., Wang, R., and Huang, R. Interactive analog layout editing with instant placement and routing legalization. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 42(3):698–711, mar 2023. ISSN 0278-0070. doi: 10.1109/TCAD.2022.3190234.

Garey, M., Johnson, D., and Stockmeyer, L. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1 (3):237–267, 1976. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(76)90059-1.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

Guo, Z. and Lin, Y. Differentiable-timing-driven global placement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, page 1315–1320, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391429. doi: 10.1145/3489517.3530486.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Hanin, B. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7 (10):992, 10 2019. ISSN 2227-7390. doi: 10.3390/math7100992.

Hao, R., Cai, Y., and Zhou, Q. Intelligent and kernelized placement: A survey. *Integration*, 86:44–50, 2022. ISSN 01679260. doi: 10.1016/j.vlsi.2022.05.002.

Hatta, K., Wakabayashi, S., and Koide, T. Solving the rectangular packing problem by an adaptive ga based on sequence-pair. volume 1999-January, pages 181–184. Institute of Electrical and Electronics Engineers Inc., 1999. ISBN 078035012X. doi: 10.1109/ASPDAC.1999.759990.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 6 2015. doi: 10.48550/arxiv.1506.05163.

Ho, C.-T., Ho, A., Fojtik, M., Kim, M., Wei, S., Li, Y., Khailany, B., and Ren, H. Nvcell 2: Routability-driven standard cell layout in advanced nodes with lattice graph routability model. In *Proceedings of the 2023 International Symposium on Physical Design*, ISPD '23, page 44–52, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9781450399784. doi: 10.1145/3569052.3578920.

Ho, C.-T., Ho, A., Fojtik, M., Kim, M., Wei, S., Li, Y., Khailany, B., and Ren, H. NVCell 2: Routability-Driven Standard Cell Layout in Advanced Nodes with Lattice Graph Routability Model. In *Proceedings of the 2023 International Symposium on Physical Design*, pages 44–52, Virtual Event USA, March 2023b. ACM. ISBN 978-1-4503-9978-4. doi: 10.1145/3569052.3578920.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Hussenot, L., Dadashi, R., Dulac-Arnold, G., Orsini, M., Jacq, A., Ferret, J., Vieillard, N., Ghasemipour, S. K. S., Girgin, S., Pietquin, O., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Friesen, A., Haroun, R., Novikov, A., Colmenarejo, S. G., Cabi, S., Gulcehre, C., Paine, T. L., Srinivasan, S., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

Holtz, C., Merrill, D. J., and Woo, M. Sa-pcb: Simulated annealing-based placement for pcb layout. `https://github.com/The-OpenROAD-Project/SA-PCB`, 2020. 2022, December 20.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8.

Hsu, M.-K., Chang, Y.-W., and Balabanov, V. Tsv-aware analytical placement for 3d ic designs. In *Proceedings of the 48th Design Automation Conference*, DAC '11, page 664–669, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306362. doi: 10.1145/2024724.2024875.

Hu, B. and Marek-Sadowska, M. Multilevel fixed-point-addition-based vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24:1188–1203, 8 2005. ISSN 02780070. doi: 10.1109/TCAD.2005.850802.

Huang, G., Hu, J., He, Y., Liu, J., Ma, M., Shen, Z., Wu, J., Xu, Y., Zhang, H., Zhong, K., Ning, X., Ma, Y., Yang, H., Yu, B., Yang, H., and Wang, Y. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 26(5), jun 2021. ISSN 1084-4309. doi: 10.1145/3451179.

Huang, Y.-H., Xie, Z., Fang, G.-Q., Yu, T.-C., Ren, H., Fang, S.-Y., Chen, Y., and Hu, J. Routability-driven macro placement with embedded cnn-based prediction model. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 180–185, 2019. doi: 10.23919/DATE.2019.8715126.

IPC. *Generic Standard on printed board design*. IPC, 2012.

Ismail, F. S., Yusof, R., and Khalid, M. Optimization of electronics component placement design on pcb using self organizing genetic algorithm (soga). *Journal of Intelligent Manufacturing*, 23:883–895, 2012. ISSN 09565515. doi: 10.1007/s10845-010-0444-x.

Jindal, T., Alpert, C. J., Hu, J., Li, Z., Nam, G.-J., and Winn, C. B. Detecting tangled logic structures in vlsi netlists. In *Proceedings of the 47th Design Automation Conference*, DAC '10, page 603–608, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300025. doi: 10.1145/1837274.1837422.

Jones, D. and Harris, i. *PCB Design Tutorial*. David Jones, Australia, 1st edition, 2004.

Kaeslin, H. *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, USA, 1st edition, 2008. ISBN 0521882672. doi: 10.5555/1817175,.

Kahng, A. B. and Wang, Q. A faster implementation of aplace. In *Proceedings of the 2006 International Symposium on Physical Design*, ISPD '06, page 218–220, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595932992. doi: 10.1145/1123008.1123057.

Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Publishing Company, Incorporated, 2nd edition, 2022. ISBN 9783030964146. doi: 10.1007/978-3-030-96415-3.

Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7:69–79, 3 1999. ISSN 1063-8210. doi: 10.1109/92.748202.

Khailany, B., Ren, H., Dai, S., Godil, S., Keller, B., Kirby, R., Klinefelter, A., Venkatesan, R., Zhang, Y., Catanzaro, B., and Dally, W. J. Accelerating chip design with machine learning. *IEEE Micro*, 40:23–32, 11 2020. ISSN 0272-1732. doi: 10.1109/MM.2020.3026231.

Kim, M.-C. and Markov, I. L. Complx: A competitive primal-dual lagrange optimization for global placement. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 747–752, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311991. doi: 10.1145/2228360.2228496.

Kim, M. C., Lee, D. J., and Markov, I. L. Simpl: An effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31:50–60, 1 2012a. ISSN 02780070. doi: 10.1109/TCAD.2011.2170567.

Kim, M.-C., Viswanathan, N., Alpert, C. J., Markov, I. L., and Ramji, S. Maple: Multilevel adaptive placement for mixed-size designs. In *Proceedings of the 2012 ACM International Symposium on International Symposium on Physical Design*, ISPD '12, page 193–200, New York, NY, USA, 2012b. Association for Computing Machinery. ISBN 9781450311670. doi: 10.1145/2160916.2160958.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 12 2014. doi: 10.48550/arxiv.1412.6980.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 11 2016. doi: 10.48550/arxiv.1611.07308.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–14, 2017.

Kirby, R., Godil, S., Roy, R., and Catanzaro, B. Congestionnet: Routing congestion prediction using deep graph neural networks. volume 2019-October, pages 217–222. IEEE, 10 2019. ISBN 978-1-7281-3915-9. doi: 10.1109/VLSI-SoC.2019.8920342.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

Kleinhans, J. M., Sigl, G., Johannes, F. M., and Antreich, K. J. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10:356–365, 1991. ISSN 19374151. doi: 10.1109/43.67789.

Kudva, P., Sullivan, A., and Dougherty, W. Metrics for structural logic synthesis. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '02, page 551–556, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 0780376072. doi: 10.1145/774572.774653.

Kullback, S. and Leibler, R. On information and sufficiency. *The Annals of Mathematical Statistics*, 22:79–86, 1951.

Lee, C. Y. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3): 346–365, 9 1961. ISSN 0367-7508. doi: 10.1109/TEC.1961.5219222.

Lee, J. Thermal placement algorithm based on heat conduction analogy. *IEEE Transactions on Components and Packaging Technologies*, 26(2):473–482, 2003. doi: 10.1109/TCAPT.2003.815091.

Li, W., Li, M., Wang, J., and Pan, D. Z. Utplacef 3.0: A parallelization framework for modern fpga global placement: (invited paper). In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, page 922–928. IEEE Press, 2017. doi: 10.1109/ICCAD.2017.8203879.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. RLlib: Abstractions for distributed reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018.

Liao, H., Zhang, W., Dong, X., Poczos, B., Shimada, K., and Burak Kara, L. A deep reinforcement learning approach for global routing. *Journal of Mechanical Design*, 142(6), 11 2020. ISSN 1050-0472. doi: 10.1115/1.4045044.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. 9 2015. doi: 10.48550/arxiv.1509.02971.

Lin, T., Chu, C., and Wu, G. Polar 3.0: An ultrafast global placement engine. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, page 520–527. IEEE Press, 2015. ISBN 9781467383899.

Lin, T.-C., Holtz, C., Yenyi, and Merrill, D. J. Printed circuit board (pcb) router. `https://github.com/The-OpenROAD-Project/PcbRouter`, 2020. 2022, December 20.

Lin, Y., Jiang, Z., Gu, J., Li, W., Dhar, S., Ren, H., Khailany, B., and Pan, D. Z. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40:748–761, 2021. ISSN 19374151. doi: 10.1109/TCAD.2020.3003843.

Lopera, D. S., Servadei, L., Kiprit, G. N., Hazra, S., Wille, R., and Ecker, W. A survey of graph neural networks for electronic design automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 8 2021. ISBN 978-1-6654-3166-8. doi: 10.1109/MLCAD52597.2021.9531070.

Lu, J., Chen, P., Chang, C.-C., Sha, L., Huang, D. J.-H., Teng, C.-C., and Cheng, C.-K. Eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. volume 20, New York, NY, USA, mar 2015a. Association for Computing Machinery. doi: 10.1145/2699873.

Lu, J., Zhuang, H., Chen, P., Chang, H., Chang, C. C., Wong, Y. C., Sha, L., Huang, D., Luo, Y., Teng, C. C., and Cheng, C. K. Eplace-ms: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34:685–698, 5 2015b. ISSN 02780070. doi: 10.1109/TCAD.2015.2391263.

Lu, J., Zhuang, H., Kang, I., Chen, P., and Cheng, C.-K. Eplace-3d: Electrostatics based placement for 3d-ics. In *Proceedings of the 2016 on International Symposium on Physical Design*, ISPD '16, page 11–18, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340397. doi: 10.1145/2872334.2872361.

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. The expressive power of neural networks: A view from the width. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6232–6240, 2017. doi: https://doi.org/10.48550/arxiv.1709.02540.

Ma, Y., He, Z., Li, W., Zhang, L., and Yu, B. Understanding graphs in eda: From shallow to deep learning. In *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, page 119–126, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370912. doi: 10.1145/3372780.3378173.

Markov, I. L., Hu, J., and Kim, M. C. Progress and challenges in vlsi placement research. *Proceedings of the IEEE*, 103: 1985–2003, 2015. ISSN 15582256. doi: 10.1109/JPROC.2015.2478963.

Merrill, D. J. *Hungry for Fully Automated Design of Embedded Systems?* University of California, San Diego, PhD thesis, 2021.

Micheli, A. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20: 498–511, 3 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2010350.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y. J., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. A graph placement methodology for fast chip design. *Nature*, 594:207–212, 2021. ISSN 14764687. doi: 10.1038/s41586-021-03544-w.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 12 2013. doi: 10.48550/arxiv.1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2 2015. ISSN 0028-0836. doi: 10.1038/nature14236.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. June 2016. arXiv:1602.01783 [cs].

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1518–1524, 1996. doi: 10.1109/43.552084.

Murphy, J. *Neural Network Fitness Function for Optimization-based Approaches to PCB Design Automation*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2020.

Naylor, W. C., Donelly, R., and Sha, L. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer, 1998.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2014–2023, New York, New York, USA, 20–22 Jun 2016. PMLR.

Ning, P., Wang, F., and Ngo, K. D. Automatic layout design for power module. *IEEE Transactions on Power Electronics*, 28: 481–487, 2013. ISSN 08858993. doi: 10.1109/TPEL.2011.2180739.

Ning, P., Li, H., Huang, Y., and Kang, Y. Review of power module automatic layout optimization methods in electric vehicle applications. *Chinese Journal of Electrical Engineering*, 6:8–24, 2020. ISSN 20961529. doi: 10.23919/CJEE.2020.000015.

Oskay, W. and Schlaepfer, E. *Open circuits*. No Starch Press, San Francisco, CA, November 2022.

Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 2609–2615. AAAI Press, 2018. ISBN 9780999241127.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

Preston, W. J. The difference between tht and smt, 7 2018. (2022, December 7).

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in reinforcement learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5402–5415. Curran Associates, Inc., 2021.

Ren, H., Nath, S., Zhang, Y., Chen, H., and Liu, M. Why are graph neural networks effective for eda problems? (invited paper). In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392174. doi: 10.1145/3508352.3561093.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. doi: 10.1037/h0042519.

Roy, J. A., Papa, D. A., Adya, S. N., Chan, H. H., Ng, A. N., Lu, J. F., and Markov, I. L. Capo: Robust and scalable open-source min-cut floorplacer. *Proceedings of the International Symposium on Physical Design*, pages 224–226, 2005.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature (London)*, 323(6088):533–536, 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. *Proceedings of the 32nd International Conference on Machine Learning*, 37:1889–1897, 07–09 Jul 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, pages 1–12, 2017. doi: 10.48550/arxiv.1707.06347.

Sechen, C. and Sangiovanni-Vincentelli, A. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20:510–522, 4 1985. ISSN 0018-9200. doi: 10.1109/JSSC.1985.1052337.

Sechen, C. and Sangiovanni-Vincentelli, A. Timberwolf3.2: A new standard cell placement and global routing package. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, DAC '86, page 432–439. IEEE Press, 1986. ISBN 0818607025.

Sigaud, O. Combining evolution and deep reinforcement learning for policy search: a survey. *ACM Transactions on Evolutionary Learning and Optimization*, 10 2022. ISSN 2688-299X. doi: 10.1145/3569096.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 9 2014. doi: 10.48550/arxiv.1409.1556.

Sperduti, A. and Starita, A. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8:714–735, 5 1997. ISSN 1045-9227. doi: 10.1109/72.572108.

Spindler, P. and Johannes, F. M. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '07, page 1226–1231, San Jose, CA, USA, 2007. EDA Consortium. ISBN 9783981080124.

Spindler, P., Schlichtmann, U., and Johannes, F. M. Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27:1398–1411, 2008. ISSN 02780070. doi: 10.1109/TCAD.2008.925783.

Stanley, K. O. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8:131–162, 6 2007. ISSN 1389-2576. doi: 10.1007/s10710-007-9028-8.

Stanley, K. and Miikkulainen, R. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762. IEEE, 2002. ISBN 0-7803-7282-4. doi: 10.1109/CEC.2002.1004508.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

Takuma Seno, M. I. d3rlpy: An offline deep reinforcement library. In *NeurIPS 2021 Offline Reinforcement Learning Workshop*. arXiv, December 2021. doi: 10.48550/arxiv.2111.03788.

Ustun, E., Deng, C., Pal, D., Li, Z., and Zhang, Z. Accurate operation delay prediction for fpga hls using graph neural networks. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415657.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 10 2017. doi: 10.48550/arxiv.1710.10903.

Veličković, P. Everything is Connected: Graph Neural Networks. January 2023. arXiv:2301.08210 [cs, stat].

Venkatesan, R., Raina, P., Zhang, Y., Zimmer, B., Dally, W. J., Emer, J., Keckler, S. W., Khailany, B., Shao, Y. S., Wang, M., Clemons, J., Dai, S., Fojtik, M., Keller, B., Klinefelter, A., and Pinckney, N. Magnet: A modular accelerator generator for neural networks. pages 1–8. IEEE, 11 2019. ISBN 978-1-7281-2350-9. doi: 10.1109/ICCAD45719.2019.8942127.

Viswanathan, N. and Chu, C. C. N. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24:722–733, 5 2005. ISSN 02780070. doi: 10.1109/TCAD.2005.846365.

Wang, M., Yang, X., and Sarrafzadeh, M. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '00, page 260–263. IEEE Press, 2000. ISBN 0780364481. doi: 10.1109/ICCAD.2000.896483.

Watkins, C. J. C. H. and Dayan, P. Q-learning. 8:279–292, 1992.

Wirtz, A. freerouting: Advanced pcb auto-router. `https://github.com/freerouting/freerouting`, 2023. 2023, June 18.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4–24, 2021. ISSN 21622388. doi: 10.1109/TNNLS.2020.2978386.

Xie, Z., Huang, Y. H., Fang, G. Q., Ren, H., Fang, S. Y., Chen, Y., and Hu, J. Routenet: Routability prediction for mixed-size designs using convolutional neural network. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2018. ISSN 10923152. doi: 10.1145/3240765.3240843.

Xie, Z., Liang, R., Xu, X., Hu, J., Duan, Y., and Chen, Y. Net2: A graph attention network method customized for pre-placement net length estimation. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ASPDAC '21, page 671–677, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450379991. doi: 10.1145/3394885.3431562.

Xu, Q., Geng, H., Chen, S., Yuan, B., Zhuo, C., Kang, Y., and Wen, X. Goodfloorplan: Graph convolutional network and reinforcement learning-based floorplanning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3492–3502, oct 2022. ISSN 0278-0070. doi: 10.1109/TCAD.2021.3131550.

Yang, S. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.

Yenyi, Lin, T.-C., Holtz, C., , and Merrill, D. J. Kicad file importer/exporter, database, and drc checker. `https://github.com/The-OpenROAD-Project/KicadParser`, 2020. 2022, December 20.

Zhang, Y., Ren, H., and Khailany, B. Grannite: Graph neural network inference for transferable power estimation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi: 10.1109/DAC18072.2020.9218643.

Zhou, Y., Ren, H., Zhang, Y., Keller, B., Khailany, B., and Zhang, Z. Primal: Power inference using machine learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367257. doi: 10.1145/3316781.3317884.

A Infrastructure and tools

Robust infrastructure and tools are essential for building large and scalable projects. In this thesis, they allowed effortless parallelisation and automation that, in turn, increased our confidence in the results and minimised human error throughout. Figure A.1 illustrates the technology stack adopted. At the lowest levels, it describes the machine used to run the experiments and the selected operating system. Four frameworks enable the development of RL models for studying the PCB component placement. These include an automated testing and report generation facility, traditional place and route tools used for generating baselines alongside which our work is compared, a purpose-built pcb library for facilitating managing circuits for training and inference and lastly PyTorch and key Python packages for constructing our RL models and developing custom environments for investigating PCB component placement.

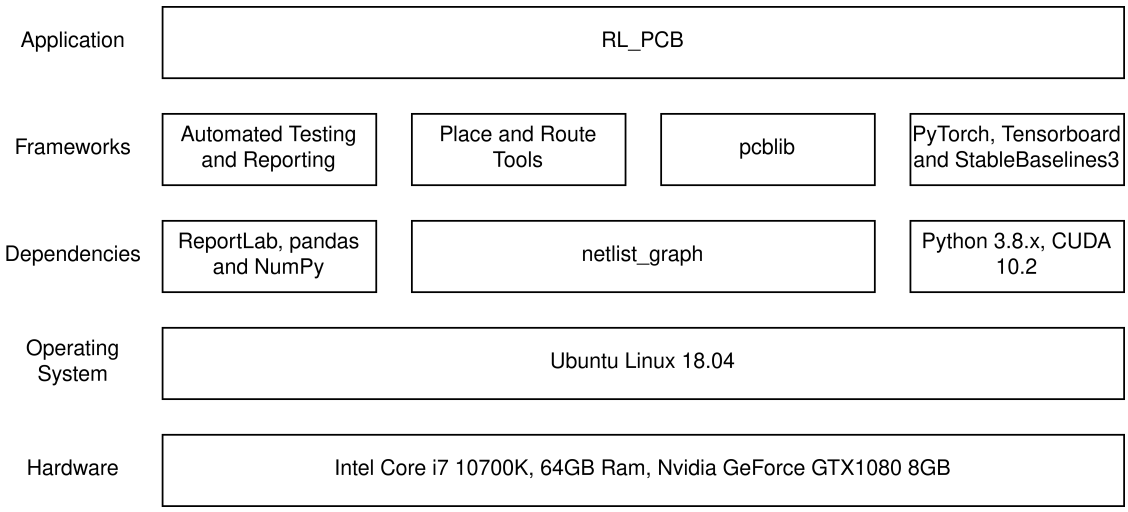


Figure A.1: Technology Stack

This section describes critical frameworks and their dependencies that we adopted or developed for this thesis. First, we discuss the selection of a CAD tool that bridges real-world PCB design and our algorithms. Next, we examine open source PCB place and

route tools adopted for the evaluation baseline and random circuit generation. Following, we present the circuit netlist representation as a graph and an encapsulating scheme that facilitates working with multiple layouts and interacting with the file system. Finally, a fully automated build and packaging system used for all software modules developed in this thesis is presented.

A.1 KiCAD PCB Design Software Suite

A CAD tool was necessary such that our automatically generated layouts could be reviewed, altered and even prepared for manufacturing in a standard way. KiCAD (Bautista et al., 2022) was selected for this project, motivated primarily by flexibility and ease of use rather than performance. KiCAD is a suite of tools for EDA that is free and open-source. It features an integrated environment for designing PCBs, including schematic capture, circuit simulation, PCB layout and manufacturing file viewer. KiCAD supports the basic features of PCB layout but lacks advanced features compared to CAD tools more commonly encountered in the industry. Examples of advanced CAD tools are Altium Designer, Cadence OrCAD schematic capture, Cadence Allegro PCB layout tools, and Pads by Mentor Graphics. One common feature that sets industrial CAD tools apart from KiCAD is the advanced simulation tools with a proven track record. The advanced simulation and layout tools do not contribute additional value to the research project, and therefore, KiCAD was selected for the following reasons:

- Being open source, the community develops supporting code to parse and manipulate the output files. Using available software infrastructure allows us to spend our time solving problem-related challenges.
- Proprietary tools require licensing, which adds an unnecessary setup overhead. Furthermore, as noted by Huang et al. (2019), educational licenses may have imposed limitations, such as an upper bound on design size.
- It is easier for other people to reproduce our work.

A.2 PCB Place and Route Tools

SA_PCB (Holtz et al., 2020) and PcbRouter (Lin et al., 2020) are open source tools for respectively placing and routing components on a PCB. SA_PCB uses SA meta-heuristic to perform component placement, which can operate directly with `.kicad_pcb` layouts,

and supports any circuit. PcbRouter uses the A* algorithm to create the associated wires between components. It supports the standard, blind and buried VIAs and thus performs multi-layer routing and advanced features to combat congestion, such as rip-up and reroute strategies. Similar to SA_PCB, it also directly operates on `.kicad_pcb` layouts. A third tool called KicadParser (Yenyi et al., 2020) is used behind the scenes to convert a `.kicad_pcb` layout into an internal representation. These tools are written in C++ and were developed due to a research project investigating a fully automated design of embedded systems and are described further in the PhD thesis by Merrill (2021).

The tool suite lacks general documentation and inline comments describing the program's operation. An effort was placed on manually digging through the source code to understand the details; however, eventually, we decided to patch the necessary functions and otherwise leave the core functionality untouched. The following is a list of changes made to the source code:

1. All three programs were augmented with a command line interface so that bash scripts could automatically call them.
2. For all programs a Makefile was created with targets for compilation, library generation and automated tests. They are version controlled and integrated in Jenkins.
3. KicadParser was enhanced to convert `.kicad_pcb` into our internal `.pcb` representation format. More details in the upcoming Appendix A.3
4. Redefinition of component outline for SA_PCB (Holtz et al., 2020).
5. Power nets could be conditionally ignored in SA_PCB (Holtz et al., 2020) and PcbRouter (Lin et al., 2020) by a command line option.

The command line feature was beneficial when generating random datasets described in Section 4.1 and when carrying out mass evaluations described in Section 4.3

Concerning update three, SA_PCB interpreted polygons on the courtyard layers as the area taken by a component on a PCB. This presented a significant problem because KiCad (Bautista et al., 2022) footprints represent a component's boundary using lines. To alter all the PCBs and draw a polygon to represent the area taken by a component was not a feasible or scalable option. Therefore we applied a modification to extract the component outline based on the lines. The challenge was that occasionally a component's footprint might have an arbitrary number of geometrical shapes drawn on the courtyard layer, including multiple quadrilaterals of different sizes or outlines defined as irregular shapes. Our method read all the lines for a given component and then extracted the largest enclosing quadrilateral to be used as its area in overlap computations.

A.3 Netlist Graph

KiCAD (Bautista et al., 2022) represents the user's design in three separate files, a schematic `.sch` file , a netlist `.net` file and layout `.kicad_pcb` file. The schematic and layout files are used for design entry and layout phases, respectively, while the netlist file propagates the design between the two stages. We are using the layout file, having an extension of `.kicad_pcb`, as input containing PCB data. It was chosen because it contains geometrical descriptions of the components and layout regions not available in the other two files.

As mentioned in Appendix A.2, the place and route programs come with little documentation, and shared very little in terms of a common structure. Therefore, it was hard to reuse the internal representation of the layout for modifications by our placement algorithms. Since this was a core function of our work, we converted the hierarchical representation of the PCB into a graph, a representation method widely employed in the literature (Huang et al., 2021; Lopera et al., 2021). Furthermore, in Section 4.1.4 we adopt GNNs for predicting circuit parameters, representing the circuit as a graph at the source will remove the need for additional pre-processing tasks. Components were represented as nodes while point-to-point connections as edges, using `node` and `edge` objects respectively. A `graph` represents the circuit netlist and comprises a vector of `node` and another of `edge`. Table A.1 describes the member variables of the object class `node`, and Table A.2 describes the member variables of the object class `edge`. Expert information, also referred to as the best historically known values of specific parameters such as net length, are also encoded in the graph. It also contains member methods for pruning sections of the graph, yielding subgraphs, computing HPWL, overlap metrics and more. Information about the layout region is stored in an object of type `board`. Currently, it contains the dimensions of the layout area. It may also integrate global design rules, such as physical layout constraints, in the future.

A.4 Internal Representation

Our algorithms operate directly on the `graph` and `board` objects. A `pcb` object was created to encapsulate `graph` and `board` objects to represent a unified PCB. Additionally, this was done to ease the moving around of many distinct layouts. An arbitrary amount of `pcb` layouts can be stored in a `pcb` vector and written to or read from a file on the operating system. This approach streamlines the process of working with multiple layouts. This simplification was beneficial since a PCB layout is frequently used in a series of tools. By representing everything in a single file, we minimise the risk of errors, maintain a clean working directory while retaining the ability to manually review the dataset.

feature name	description
id; name	A unique numerical instance id; Human readable designator.
size	Component size in mm.
position	Component coordinate position (x,y) in mm.
orientation	Orientation in degrees.
layer	Specifies whether the component is on top or bottom layer (Always top layer).
pin count	Number of pins / pads the component has.
locked	Specifies whether the component is movable.
type	Component type; e.g. capacitor, resistor
function	Component function within the circuit; e.g. decoupling, filtering; pull-up current limiting.

Table A.1: Description of node member variables in circuit netlist.

feature name	description
id; name	A unique numerical net id; Human readable net name.
power rail	Indicates whether the type of net, generic or power related.
pad size	Size of source and destination pins / pads.
pad position	Position of source and destination pins / pads.

Table A.2: Description of edge member variables in circuit netlist.

The information is encoded hierarchically, with constituents being enclosed within placeholders. For example, the graph's nodes are enclosed within `begin nodes` and `end nodes`. This encoding facilitates parsing while visually apparent for manual interpretation. For troubleshooting and traceability, all PCBs are logged with their name and parent name if they are a subset of another PCB, UNIX timestamp, HPWL and overlap metrics. The text file in snippet A.1 demonstrates the constituents of a simple `.pcb` file.

```
filename=<abs-path>/opt_testing_10.pcb
timestamp=1665423047
pcb begin
    .kicad_pcb=555_timer.kicad_pcb
    timestamp=1664918461
    graph begin
        hpwl=34.99500100
        nodes begin
            <id>,<designator>,<width>,<height>,<x>,<y>,<orientation> ...
            0,C3,3.3,1.46,116.3,94.2,0,0,0,2,2,0,-1
            1,LED2,3.35,1.85,104.29,95.72,90,0,0,2,2,0,-1
            2,R2,3.7,1.9,115.55,88.74,270,0,0,2,2,0,-1
            3,R3,3.7,1.9,116.45,91.8,180,0,0,2,2,0,-1
```

```
            4,R4,3.7,1.9,104.26,91.65,270,0,0,2,2,0,-1
            5,U1,7.4,5.4,110,90.01,0,0,1,8,8,0,-1
        nodes end
        optimals begin
            <id>,<designator>,<EW>,<HPWL>
            0,C3,1.70655,23.2538
            1,LED2,1.89853,11.8441
            2,R2,1.75134,16.3314
            3,R3,3.71927,19.6108
            4,R4,5.1408,5.89343
            5,U1,1e+06,1e+06
        optimals end
        edges begin
            0,1,2,1.075,0.95,0.8625,0,0,1,0,1,0.95,0.95,-0.75,0,0,1,GND,1
            0,1,2,1.075,0.95,0.8625,0,0,5,0,1,1.95,0.6,-2.475,-1.905,1,1,GND
    ,1
            1,0,1,0.95,0.95,-0.75,0,0,5,0,1,1.95,0.6,-2.475,-1.905,1,1,GND,1
            0,0,1,1.075,0.95,-0.8625,0,0,3,1,2,1.2,1.4,1,0,0,2,"Net-(C3-Pad1)
    ",0
            0,0,1,1.075,0.95,-0.8625,0,0,5,1,2,1.95,0.6,-2.475,-0.635,1,2,"
    Net-(C3-Pad1)",0
            0,0,1,1.075,0.95,-0.8625,0,0,5,5,6,1.95,0.6,2.475,0.635,1,2,"Net
    -(C3-Pad1)",0
            3,1,2,1.2,1.4,1,0,0,5,1,2,1.95,0.6,-2.475,-0.635,1,2,"Net-(C3-
    Pad1)",0
            3,1,2,1.2,1.4,1,0,0,5,5,6,1.95,0.6,2.475,0.635,1,2,"Net-(C3-Pad1)
    ",0
            5,1,2,1.95,0.6,-2.475,-0.635,1,5,5,6,1.95,0.6,2.475,0.635,1,2,"
    Net-(C3-Pad1)",0
            1,1,2,0.95,0.95,0.75,0,0,4,1,2,1.2,1.4,1,0,0,3,"Net-(LED2-Pad2)
    ",0
            4,0,1,1.2,1.4,-1,0,0,5,2,3,1.95,0.6,-2.475,0.635,1,4,"Net-(R4-
    Pad1)",0
            2,1,2,1.2,1.4,1,0,0,3,0,1,1.2,1.4,-1,0,0,5,"Net-(R2-Pad2)",0
            2,1,2,1.2,1.4,1,0,0,5,6,7,1.95,0.6,2.475,-0.635,1,5,"Net-(R2-Pad2
    )",0
            3,0,1,1.2,1.4,-1,0,0,5,6,7,1.95,0.6,2.475,-0.635,1,5,"Net-(R2-
    Pad2)",0
            2,0,1,1.2,1.4,-1,0,0,5,3,4,1.95,0.6,-2.475,1.905,1,6,+3V3,2
            2,0,1,1.2,1.4,-1,0,0,5,7,8,1.95,0.6,2.475,-1.905,1,6,+3V3,2
            5,3,4,1.95,0.6,-2.475,1.905,1,5,7,8,1.95,0.6,2.475,-1.905,1,6,+3
    V3,2
        edges end
    graph end
    board begin
```

```
47          bb_min_x,100.00000000
48          bb_min_y,80.00000000
49          bb_max_x,120.00000000
50          bb_max_y,100.00000000
51      board end
52  pcb end
53  pcb begin
54  ...
55  pcb end
```

Listing A.1: Example of a PCB file

A.5 Automated Builds

As stated earlier, the `graph` and `board` objects encapsulate the foundational methods leveraged by higher-level placement algorithms. The `pcb` objects encapsulate these constituents and provide file input-output methods for easy handling. They are written in C++ for performance and exposed to Python through SWIG (Beazley, 1996).

Figure A.2 illustrates a typical build process. Static libraries are compiled to be used by C++ tests and tools dependent on them. Dynamic libraries are generated as a requirement of SWIG for exposing the methods in Python. SWIG generates Python wrappers based on the libraries' header files. The wrapper and dynamic libraries are packaged into a wheels

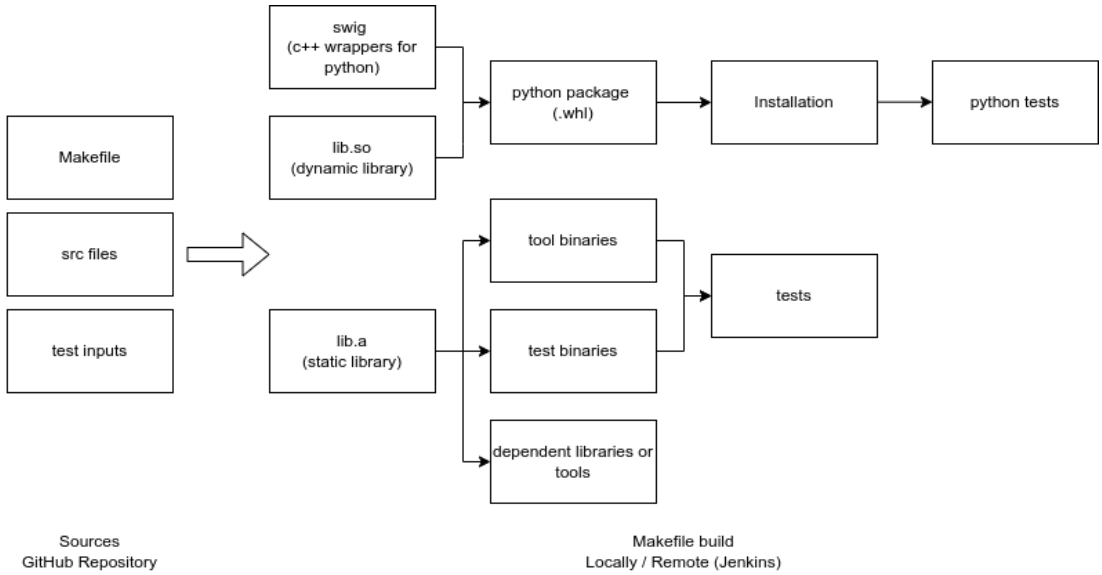


Figure A.2: Illustration of the automated building and testing the C++ source code via the bundled Makefiles and particular test cases.

(`.whl`) package to be installed and used across projects. Arising from dependencies across different tools, automated tests were included to catch errors at an early stage. However, they attempt to catch common human-related errors. The build process was carried out using Jenkins, a continuous integration tool, on a virtual machine. A code push to GitHub automatically triggered the build and test processes.

A.6 RL Framework

Stable Baselines3 (Raffin et al., 2021) is an RL framework that makes cutting-edge algorithms accessible with a few lines of code while allowing flexibility for custom implementations. Furthermore, a unified interface to multiple RL algorithms and excellent documentation are desired features for this research project. Competing frameworks include RLlib Liang et al. (2018), which has commercial backing but is intended for massive scaling across remote machines and not oriented for small projects. d3rply (Takuma Seno, 2021) is another framework; however, it uses offline learning where it trains on previously collected data. Acme Hoffman et al. (2020) is developed by the research scientist at DeepMind and is intended for algorithm development. Stable baseline3 satisfied our practical requirements for using distinct algorithms through a unified interface, reasonable flexibility, and practical features such as Tensorboard (Abadi et al., 2015) interfacing and

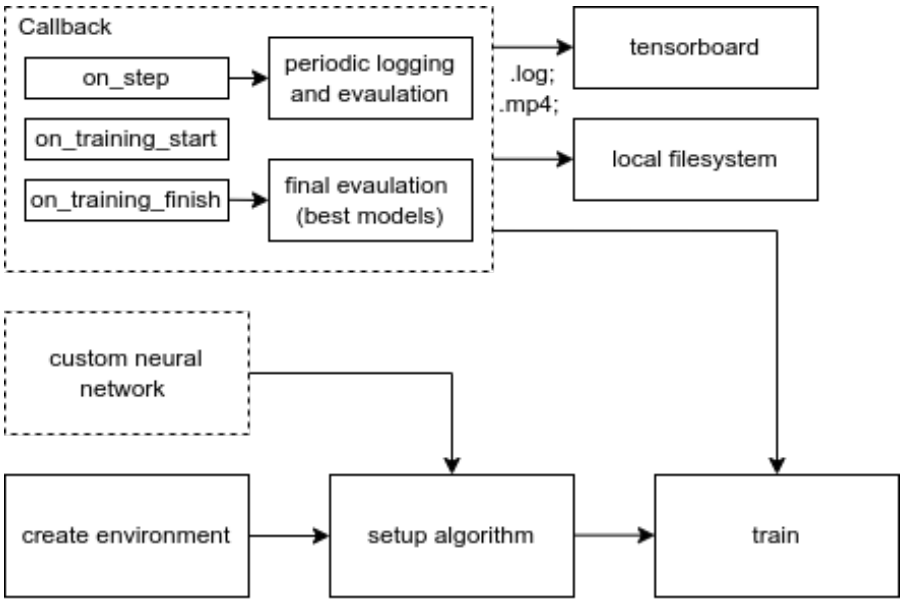


Figure A.3: Depiction of the RL training framework. The process tracking information is detailed via callbacks as well as the option to specify a custom neural architecture.

vectorised environments. Hence we chose it as the framework for this research project.

In a Stable Baselines3 (Raffin et al., 2021) standard flow, the user sets up an RL algorithm (e.g. TD3 (Fujimoto et al., 2018)) by specifying the algorithm's hyperparameters and passes a dictionary containing the neural network configuration. A callback function is optionally set up to control and monitor the learning process. Since our custom environment returns the observation as a Python dictionary, we bypass the default neural network architectures and derive a custom model from the base class the framework provides. Furthermore, we override the default callback to monitor and log a run's progress while periodically evaluating the agent using Tensorboard (Abadi et al., 2015). Figure A.3 illustrates the setup of a single training run. The dashed boxes show the methods that are overridden by our custom functions. Furthermore, the callback identifies methods invoked at every training step and the start and finish of a training run. It also accompanies a subset of tasks that may be carried out.

A.7 Reproducibility of Results

The previous section highlighted the importance of carrying out multiple runs for a single experiment and averaging them. Quantitative results are significant in RL (Raffin et al., 2021), and a fair comparison requires their execution under identical conditions. With this in mind, the experiments in this project may be seeded completely randomly or based on a known seed. The latter was adopted throughout the thesis, with a value of 99.

It becomes complicated when forking multiple processes and parallelising runs because the operating system has an element of stochasticity. For this reason, the seed values are generated in the main process before forking multiples run. They are passed individually to each run to be used as a seed in its pseudo-random number generators. Vectorised environments are seeded based on the pseudo-random number generator within the run. Runs and their vectorised environments are initialised for a single experiment with unique seed values. However, these sets of seed values are standard across experiments allowing, as mentioned earlier, a fair comparison. Evaluation for all experiments, including their runs, is based on a different non-vectorised environment always seeded with the same values. In other words, the evaluation environment is constant.

It should be noted that neural architecture search (Section 4.2.7.2) is parallelised within the Optuna (Akiba et al., 2019) framework. Optuna uses random seeds for individual parallel runs and suggests a sequential operation when deterministic results are desired. An exception is made here in the interest of time since parallelisation reduces the run time sixfold from as much as twelve days to as little as two days.

B Datasets

This section provides supplementary information about the datasets used in this thesis which may be particularly helpful for replicating parts of this work. The pool of circuits from which all datasets are derived is described in Table B.1.

ID	Layout Name	Central Component	Layout Type	# Components	Dimensions (mm)
01	555_timer	NE555	Mixed-Signal	5	20x20
02	audio_preamp	TL072	Analogue	13	20x20
03	audio_rx	CS5343	Mixed-Signal	15	20x20
04	audio_tx	CS4344	Mixed-Signal	15	20x20
05	butterworth_lpf	TL071	Analogue	4	20x20
06	differential_amplifier	TL074	Analogue	7	20x20
07	diff_gyro_afe	TL074	Analogue	13	20x20
08	ftdi	FT4232HQ	Analogue	21	30x30
09	iic_gpio_expander	PCA9500PW	Digital	11	20x20
10	lan8720	LAN8720	Mixed-Signal	16	30x30
11	led0	BC847	Analogue	2	20x20
12	led1	BC847	Analogue	4	20x20
13	max9744	MAX9744	Mixed-Signal	28	30x30
14	mcp73871	MCP73871	Mixed-Signal	11	20x20
14	neo_m9n	NEO_M9N	Mixed-Signal	8	30x30
16	notch_filter	TL071	Analogue	6	20x20
17	notch_filter2	TL072	Analogue	8	20x20
18	PModBoard	MPU6050	Digital	4	20x20
19	spi_flash	W25Q32	Digital	6	20x20
20	tc_logger_max232	MAX3232	Analogue	4	20x20
21	tc_logger_max31856	MAX31856	Mixed-Signal	7	20x20
22	tc_logger_mcu	P18F26Q10	Digital	3	20x20
23	tc_logger_silabs	CP2102	Digital	7	20x20
24	usb_controller	MAX3421	Mixed-Signal	7	20x20
25	usb_host_cli	USB3300	Mixed-Signal	8	20x20
26	voltage_datalogger_adc0	MCP3564	Mixed-Signal	8	20x20
27	voltage_datalogger_adc1	MCP3564	Mixed-Signal	16	30x30
28	voltage_datalogger_adc2	TL071	Analogue	2	20x20
29	voltage_datalogger_afe	TL074	Analogue	8	20x20
30	w5500	W5500	Mixed-Signal	18	30x30

Table B.1: Enumeration of all circuits in the dataset.

Field	Type	Description
Nodes	Ordered list	List of node attributes ordered by component id. Constituents as described in Table
Edges	List	Adjacency list
Edge Attributes	List	Edge attributes
Board size (x)	Double	Board horizontal dimension (mm)
Board size (y)	Double	Board vertical dimension (mm)
Completion	Double	Layout completion as a fraction of the parent layout
Layout ID	Integer	Derived from, but tracable to the parent
Layout Name	String	Identical to the parent
Iterations	Integer	Number of placement optimisations with Holtz et al. (2020) simulated annealing
Overlap	Double	Overlap area between components
HPWL	Double	Approximate wirelength
RUDY wirelength	Double	Approximate wirelength based on
Routed wirelength	Double	Routed wirelength (optional)
Routed vias	Integer	Number of vias introduced by Lin et al. (2020) A* star router (optional)

Table B.2: Enumeration of attributes associated with a circuit netlist.

B.1 Constructive Placement Dataset

This section contains supplementary information about the dataset generation described in Section 4.1.4.2 as part of the wirelength prediction task. Table B.2 describes all the

Parameter	Value	Description
Random captures	4096	Number of partial layout to generate
Optimisations	2	Optimise each partial layout twice
Optimisation multiplier	2	Number of layout components multiplied by this value provides the upper bound for simulated annealing iterations.
Minimum iterations	8	Absolute lower bound on simulated annealing iterations
Maximum iterations	2048	Absolute upper bound on simulated annealing iterations
Route	True	Routed wirelength targets are generated
Rip up and reroute iterations	5	Rip up congested regions and reroute to improve wirelength
Layer change cost	100	Cost associated with the introduction of a via
Unique base layouts	20	Further details available in Table

Table B.3: Configuration for automatic dataset generation. The subsequent dataset was employed in graph-level wirelength prediction task.

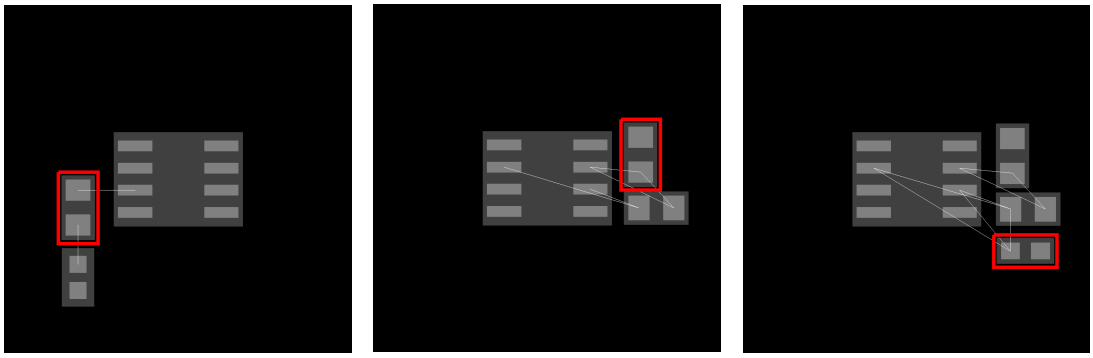
Layout name	Train / Test Dataset # Layouts	% Layouts	Unseen Dataset # Layouts	% Layouts
555_timer	758	9.25%	0	0%
led_1	220	2.69%	0	0%
differential_amplifier	308	3.76%	0	0%
voltage_datalogger_afe	344	4.2%	0	0%
iic_gpio_expander	498	6.08%	0	0%
usb_controller	0	0%	134	26.17
tc_logger_mcu	136	1.66%	0	0%
tc_logger_silabs	0	0%	156	30.47
PModBoard	0	0%	94	18.36
notch_filter	268	3.27%	0	0%
tc_logger_max232	192	2.34%	0	0%
butterworth_lpf	164	2.0%	0	0%
spi_flash	272	3.32%	0	0%
tc_logger_max31856	308	3.76%	0	0%
usb_host_cli	362	4.42%	0	0%
voltage_datalogger_adc0	394	4.81%	0	0%
notch_filter2	0	0%	128	25
audio_preamp	520	6.35%	0	0%
audio_rx_s	626	7.64%	0	0%
audio_rx_m	606	7.4%	0	0%
audio_tx_s	610	7.45%	0	0%
audio_tx_m	564	6.88%	0	0%
diff_gyro_afe	550	6.71%	0	0%
mcp73871	492	6.01%	0	0%
Total	8192	100%	512	100%

Table B.4: Detailed dataset constituents used for wirelength prediction.

elements associated with each graph data point in the dataset, while Table B.3 contains the configuration used for randomly generating it. Furthermore, Table B.4 accumulates the number of individual circuits used in the training and testing dataset comprised of 8192 random circuits and the unseen dataset comprised of 512 random circuits. The percentage contribution is also listed and indicates that larger circuits were sampled more often than smaller ones.

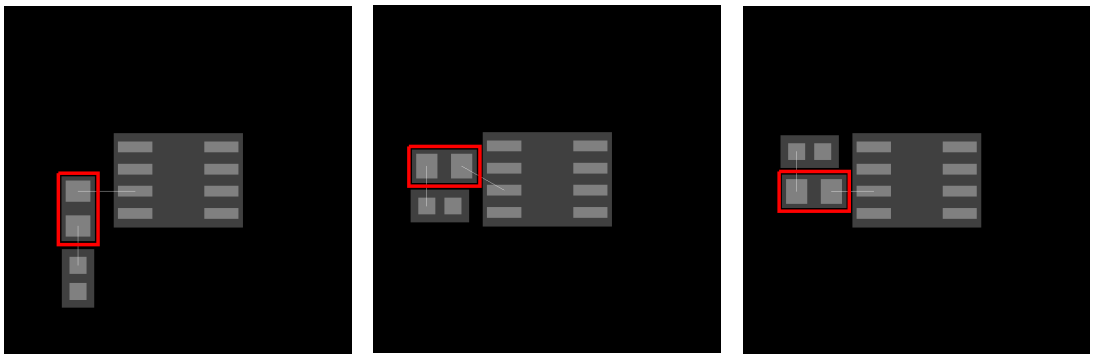
B.2 Single-Component Iterative Placement Dataset

The single-component dataset was derived from a single circuit. Concerning Table B.1, the layout name is 555_timer. Dataset D_1 comprises three circuits depicted in Figure B.1

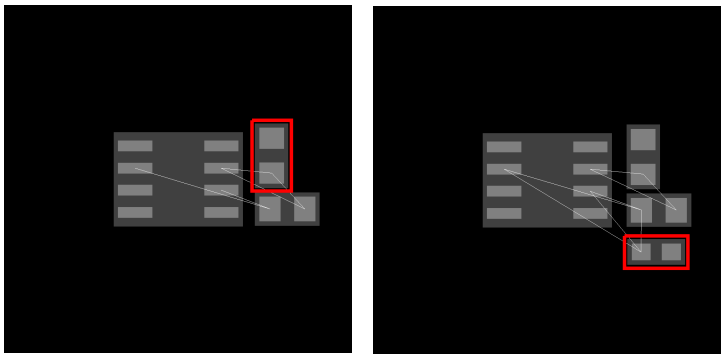


(a) D_{1a} (b) D_{1b} (c) D_{1c}

Figure B.1: Constituents of the single-component dataset, D_1 used for studying policy learning on individual circuits. The component enclosed within a red bounding box is movable and will be controled by the policy.



(a) D_{2a} (b) D_{2b} (c) D_{2c}



(d) D_{2d} (e) D_{2e}

Figure B.2: Constituents of the multi-component dataset, D_2 used for studying generalisation. The component enclosed within a red bounding box is movable and will be controlled by the policy.

used in distinct training runs. Dataset D_2 illustrated in Figure B.2 contains five circuits simultaneously used in a single training run. The red boxes highlight the single moveable component, and the remaining circuits were locked in KiCad (Bautista et al., 2022). Section 4.2.6 describes the usage of these datasets in further detail.

B.3 Multi-Component Iterative Placement Dataset

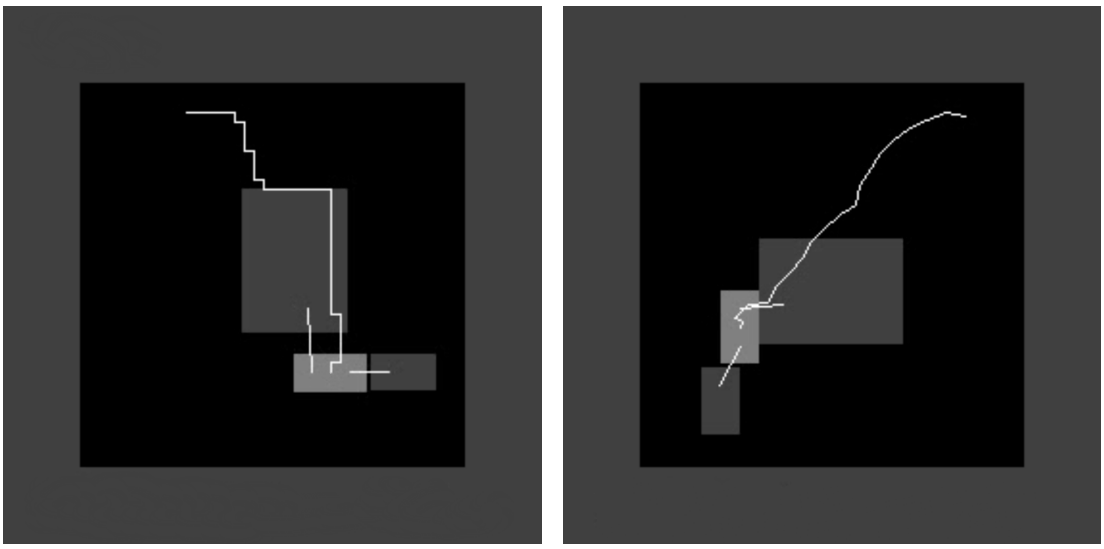
The multi-component dataset was introduced in Section 4.3.4. Here were provide supplementary information about its constituents. Table B.5 presents all the nine circuits in the dataset selected from Table B.1. Six circuits comprise the training dataset M_{Tx} while the remaining make up the testing dataset denoted by M_{Ux}. The latter was exclusively used for evaluation purposes to assess the generalisation capabilities of our approach.

Layout name	Handle	Training / Unseen	# Components	Mixed size
555_timer	M_{T0}	Training	5	Y
PModBoard	M_{T1}	Training	4	N
tc_logger_max232	M_{T2}	Training	4	N
tc_logger_max31856	M_{T3}	Training	7	Y
tc_logger_mcu	M_{T4}	Training	3	N
tc_logger_silabs	M_{T5}	Training	7	Y
voltage_datalogger_adc0	M_{U0}	Testing	8	Y
voltage_datalogger_adc2	M_{U1}	Testing	2	N
voltage_datalogger_afe	M_{U2}	Testing	8	N

Table B.5: Multi-component dataset employed fro training and testing.

C Supplementary Experiments

This section contains supplementary experiments performed to develop an understanding of particular aspects of our setup. Many of these tests were performed to replace assumptions with data-driven decisions. Specifically, they contain pre-preliminary test setups used to validate our environments, additional analysis of optimisation runs and quantitative experiments that draw relationships between parameters of interest.



(a) Example using a discrete action space (b) Example using a continuous action space

Figure C.1: Illustration of expected policy behaviour with discrete and continuous action spaces.

C.1 Expectations from Single-Component Iterative Placement

Our RL agent is provided with three pieces of information, a description of its surroundings, a general direction pointing towards the target region, and its position and attitude. These represent the observation derived from problem features. The policy learns to carry out actions that guide the agent through obstacles and ultimately place the component in the circuit. Figure C.1(a) and C.1(b) respectively illustrate the trajectory taken by a trained agent using a discrete and continuous action space.

C.2 Single-Component Placement with Discrete Actions

Supplementary experiments investigating the effects of particular environment parameters, such as episode length and step size, are presented in this section. We also study the impact of random initialisation on training performance because the discrete nature of the action space requires varying steps for travelling a certain distance depending on the target angle. Lastly, we reveal additional detail on the results of the combined hyperparameter and NAS.

C.2.1 Quantitative Analysis of Episode and Step Length

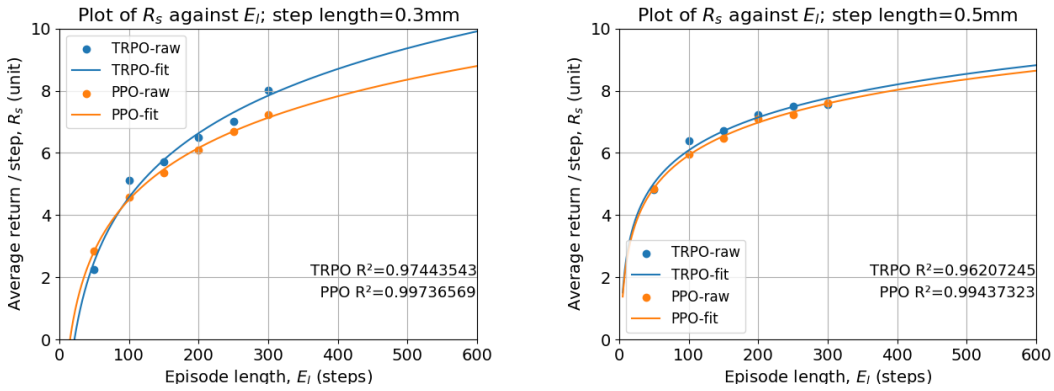
A discrete action space limits movement to defined step lengths, resulting in a positional error based on initial conditions and the step length, which can reach a maximum error equivalent to the step length. This can affect training performance: choosing a small step size reduces positional error but prolongs episodes with greater variation, while selecting a large step size results in shorter episodes but potentially larger positional errors.

The length of an episode may also affect the performance of the training process, as a trained agent is expected to complete a fixed number of steps on average within a predetermined layout size. If the agent spends most of its episode time near the optimal location, the collected data points will be very similar and will not be very useful for training. Some deviation in the intermediate steps required to reach the optimal location will be present due to random initial conditions. Consequently, the episode length can impact both the duration and quality of the learned policy, and thus it is worth investigating.

We attempt to co-optimise the episode and step length by varying the episode length from 50 to 300 steps in increments of 50 for three different step lengths. The experimental setup is summarised in Table C.1. Since we anticipate the agent's reward to level

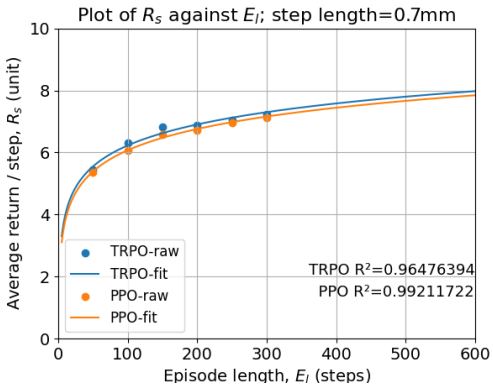
off once it approaches the optimal location, we fit a logarithmic function to the collected data and select the best parameters.

C.2.1.1 Experimental Results



(a) step length = 0.3mm

(b) step length = 0.5mm



(c) step length = 0.7mm

Figure C.2: Relationship between return-per-step and episode length using a discrete action space for three step length values.

Figure C.2 illustrates the results for sweeping the episode length for three step sizes ranging from 0.3 to 0.7mm. Since we expected a logarithmic relationship, we fit a log-

Parameter	Value	Step	Description
episode steps	50 - 300	50	The maximum number of steps allowed in the environment
step length	0.3 - 0.7	0.2	The discrete step length in mm

Table C.1: Configurations for episode and step length experiments.

arithmic function to our data points and extracted a log function with an R^2 coefficient exceeding 0.96 in all cases. A high R^2 coefficient indicates that our data highly correlates with the interpolated logarithmic function.

The trade-off imposed by the step size is one of accuracy against movement distance and is evident between the two extremes depicted in FiguresC.2(a) and C.2(c). The former requires significantly more steps for the average return per step to stabilise, while the opposite is observed in the latter. The average return per step is expected to be higher for smaller step sizes indicating that more accurate positioning is possible. The final positional error arising from the quantised action can contribute a significant error and is exacerbated by the non-linear nature of the tangent function since it asymptotically reaches infinity as its dependent variable approaches $\frac{\pi}{2}$. Therefore minor errors close to $\frac{\pi}{2}$ can have a severe impact on the average return.

The impact of the episode length is less significant, and setting it to a small one may result in the policy not converging, while setting it high will result in unnecessarily long training runs. The latter may also impact performance by overwhelming the replay buffer with highly correlated data in which the agent remains relatively still. The in-between step size of 0.5 achieves balance when coupled with an episode length between 150 and 250 steps. We adopted an episode length of 200 in our tests; however, additional experiments with an episode length of 100 converged consistently.

C.2.1.2 Effect of Initialisation

Random initial conditions may affect the training performance due to requiring a varying number of steps to the target. This is exacerbated for discrete action spaces by the fact that the agent can only move at an angle that is an integer multiple of 90 degrees. Figure C.3 depicts the difference in movement between a discrete and continuous action space. For the former, the component must first move along the opposite side, followed by the adjacent side, and the latter has to move along the hypotenuse. By analogy, when the component is initialised at the worst-case angle of 45 degrees relative to the optimal location, a discrete action space requires 41.42% more steps to reach the goal. We investigate this variability's impact on training performance by fixing the initialisation condition to a handful of starting locations. The distance to the optimal goal region is fixed at 6mm at an angle of 30 degrees. Since the data augmentation module is retained, there will still be variations at every episode step, preventing this from being a trivial experiment.

C.2.1.3 Experimental Results

Throughout the training process, two features introduce variety across episodes such that the policy may learn the fundamentals and generalise to differing setups. These are the data augmentation module discussed in Section 4.2.6.2 and random component initialisation. In this experiment, we fix the initialisation at a fixed distance and angle to the goal distance while retaining the data augmentation feature. Doing so sets the number of steps the agent requires to reach the goal throughout the episode.

The impact of this setup condition resulted in reduced standard deviation throughout the training process, as is evident from Figure C.4 and numerically in Table C.2. For fixed initialisation, the average tends to suffer slightly, ranging from 0.38% for TRPO using reward R_{2a} to 3.27% for TRPO guided by reward R_{2b}. The exception is PPO trained with rewards signal R_{2b}, in which a 0.267% improvement is registered. The standard deviation, on average, drops by 60%. Another noteworthy observation is that fixed initialisation leads to slightly faster training, which is implied by the steeper learning average return curve. This may be attributed to less variation experienced throughout training and a more straightforward, albeit not necessarily better, path to success.

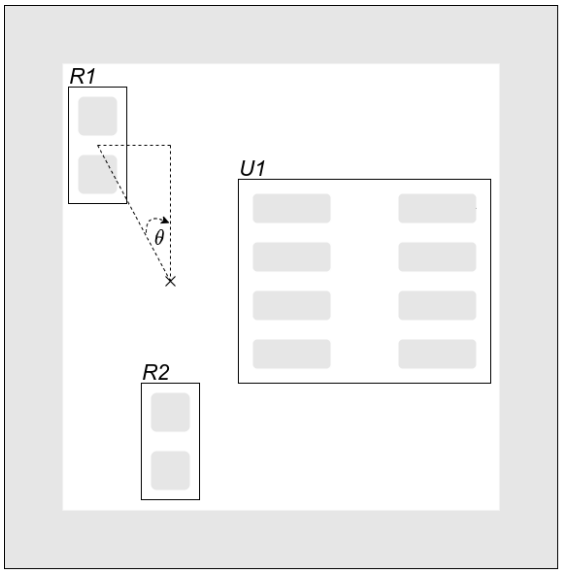


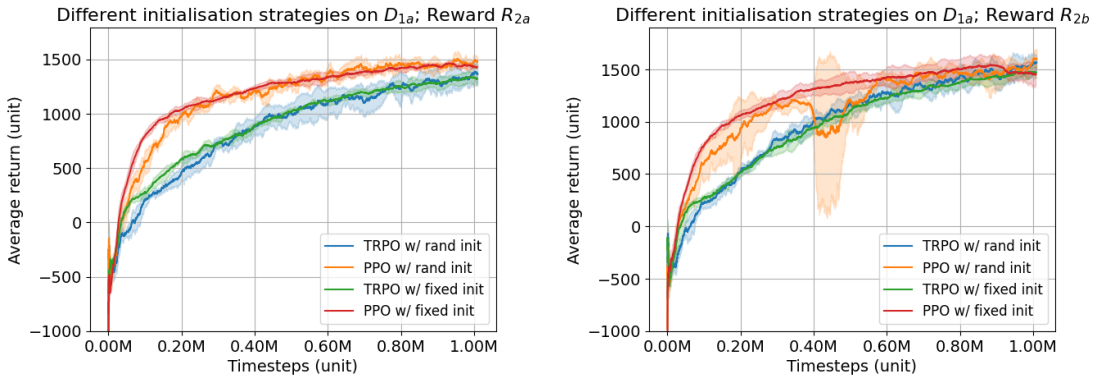
Figure C.3: Illustration of random initialisation influencing the learning process. The number of discrete steps for a fixed target distance is conditional on the target angle θ.

	reward signal R_{2a}		reward signal R_{2b}	
	Random initialisation	Fixed initialisation	Random initialisation	Fixed initialisation
TRPO	1300.57 $\pm$ 380.16	1295.33 $\pm$ 155.66	1484.20 $\pm$ 382.19	1437.66 $\pm$ 142.93
PPO	1455.22 $\pm$ 358.31	1438.76 $\pm$ 135.07	1494.93 $\pm$ 419.06	1498.24 $\pm$ 180.51

Table C.2: Average return contrasting the impact of fixed and random initialisation on learning performance. A discrete action space and reward R_{2x} were considered.

C.2.1.4 Neural Architecture Search

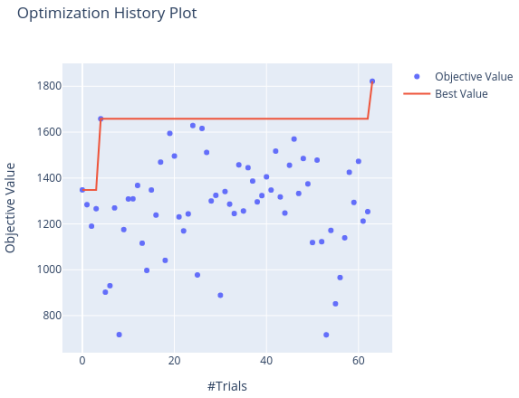
The slice plot in Figures C.5(c) and C.6(c) illustrate how individual parameters affect the objective value. As multiple trials are performed, parameter sets can be identified. For example, concerning TRPO, a tanh activation function, combined with a policy network of 2-3 layers and a value network of arbitrary size, will likely be a good selection. Concerning PPO, the number of policy layers is deemed highly important according to Figure C.6(b), likely due to the inability to surpass a return value of 1300 when employing a single layer as indicated in the slice plot in Figure C.6(c). Similar to TRPO, PPO is insensitive to the number of value layers and the activation function. The optimisation histories for TRPO in Figure C.5(a) and PPO in Figure C.6(a) show that the best architecture yields a return that deviates from the rest. A scenario that is more severe in TRPO and may be attributed to the optimisation process' dependence on the random seed. The implications of random initialisation and reproducibility of results were discussed in Appendix



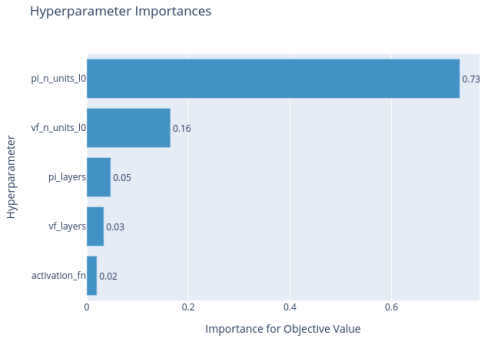
(a) Reward signal R_{2a}

(b) Reward signal R_{2b}

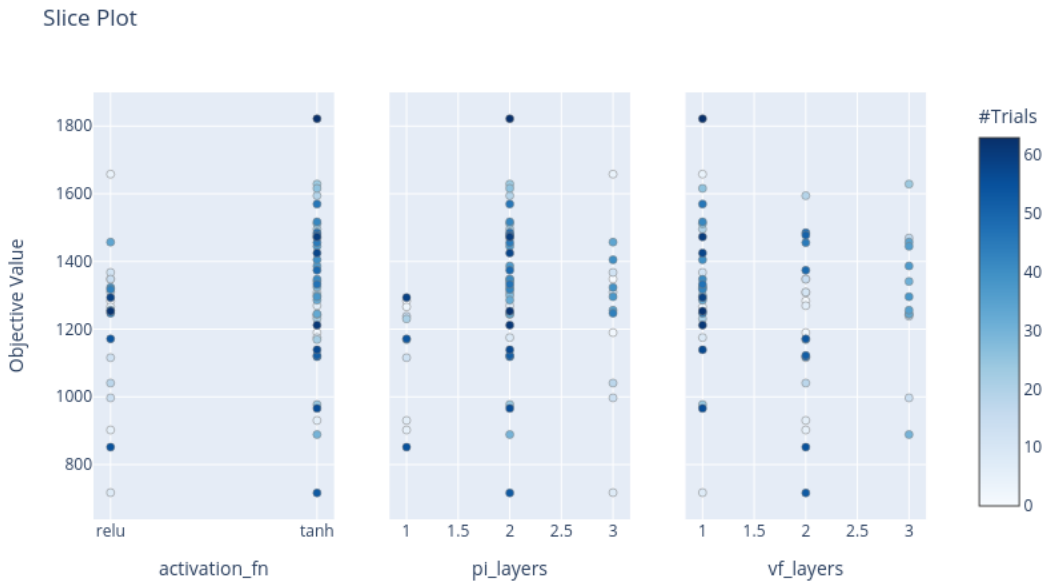
Figure C.4: Average return for training with fixed and random initialisation with a discrete action space and reward signal R_{2x}. Random initialisation introduced a notable increase in standard deviation.



(a) Optimisation history



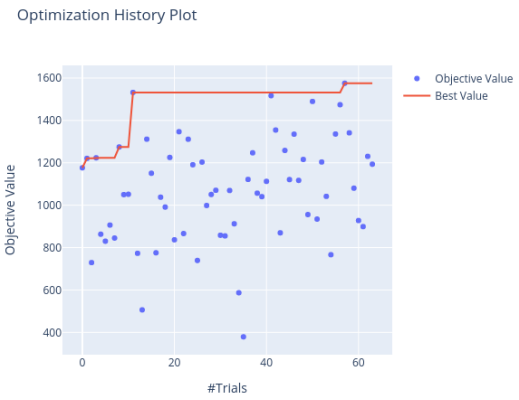
(b) Hyperparameter importance



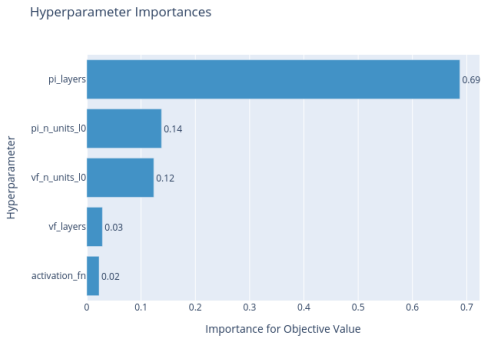
(c) Slice chart correlating activation function and network depth with objective cost

Figure C.5: NAS results for TRPO with a discrete action space and R_{2x}

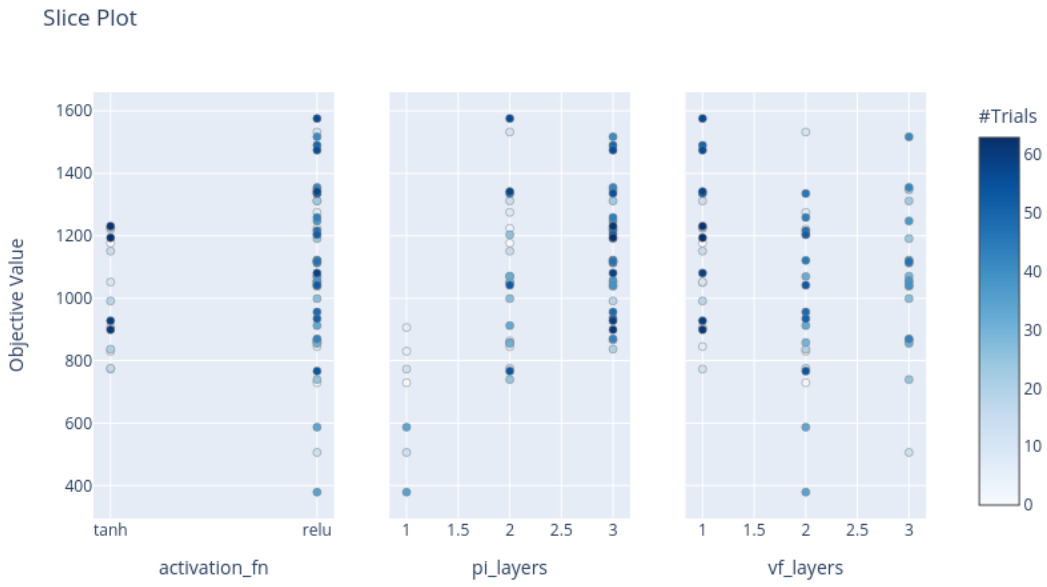
A.7. Since, for practical reasons, parallelisation was necessary, one way to reduce the risk of an outlier impacting the results is to average the trial return over multiple runs. Such an approach would incur a computational penalty albeit still result in higher throughput than a sequential approach while attenuating the impact of a lucky run.



(a) Optimisation history



(b) Hyperparameter importance



(c) Slice chart correlating activation function and network depth with objective cost

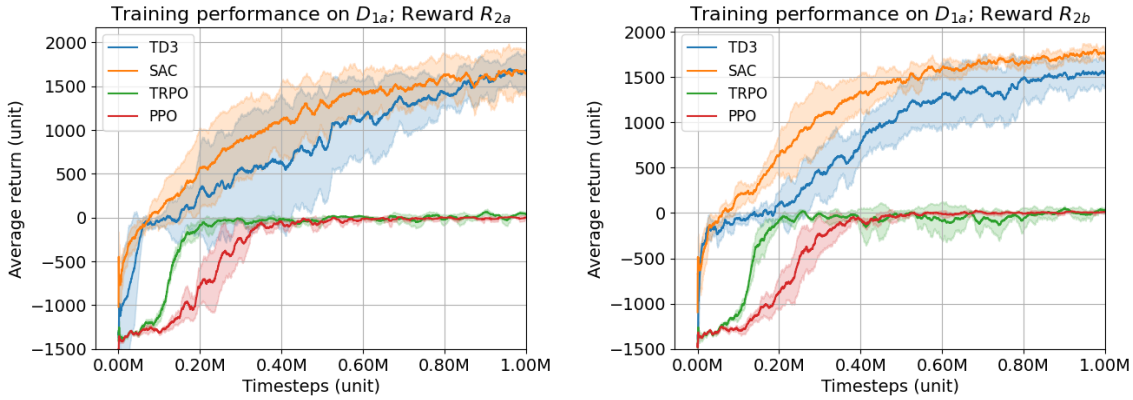
Figure C.6: NAS results for PPO with a discrete action space and R_{2x}

C.3 Single-Component Placement with Continuous Actions

This section presents further experiments related to the single-component setup with continuous action spaces. In particular, we present preliminary experiments illustrating the limitations of policy optimisation (TRPO and PPO) and the success of actor-critic (TD3 and SAC) algorithms on this setup. We also sweep a range of episode lengths to identify a practical value that achieves a good trade-off in terms of average return per step.

C.3.1 Preliminary Experiments

Continuous action spaces tend to require more training effort when compared to policies with discrete action spaces. A control experiment was performed to establish a baseline prior to carrying out extensive experiment runs. Using the Stable Baslines3 (Raffin et al., 2021) default setup, we found that on-policy algorithms fail to learn a policy for placing a single moveable component in a fixed circuit. By contrast, off-policy algorithms performed exceptionally well. Figure C.7 illustrates the average return for all four algorithms under consideration as a function of absolute steps carried out in the environment and are further summarised numerically in Table C.3.



(a) Training guided by reward signal R_{2a}

(b) Training guided by reward signal R_{2b}

Figure C.7: Average return for the considered RL algorithms on continuous action spaces. TD3 and SAC actor-critic algorithms are successful on the task, in contrast with on-policy TRPO and PPO.

According to Figure C.7 the TRPO and PPO are initially assigned large negative rewards that gradually decrease and settle around zero. This behaviour is motivated by the avoidance of the early termination penalty. During the initial episode, the policy often

	Reward R_{2a}	Reward R_{2b}
TRPO	13.38 $\pm$ 189.70	-9.13 $\pm$ 249.18
PPO	-3.60 $\pm$ 114.74	8.32 $\pm$ 54.11
TD3	1594.00 $\pm$ 515.31	1516.60 $\pm$ 447.67
SAC	1615.82 $\pm$ 503.37	1734.97 $\pm$ 392.44

Table C.3: Average return for optimised experiment configurations with a continuous action space.

drives the component into the padding and outside the board region. When such a situation happens, the episode terminates early, and the agent is penalised proportionally to the remaining steps. Therefore by remaining stationary, the agent avoids the penalty altogether, leading to a relatively higher return, albeit useless. On-policy algorithms collect small batches of data points through trial-and-error interaction with the environment, which they use and discard. If an exemplary sequence of actions is hard to find, the agent may not be able to generate sufficient data points of high quality to learn. Therefore, the policy may not learn due to its inability to collect high-quality data as a result of being stupid, which is a catch-22 situation. On-policy algorithms were successful with a discrete action space containing a set of six possibilities, but with a continuous action space, the possibilities are infinite and thus more challenging. Off-policy algorithms are less susceptible to this phenomenon because they store data points in a replay buffer which can be reused over an extended time.

C.3.2 Quantitative Analysis of Episode Length

The experiment is identical to that described in Appendix C.2.1 albeit concerning TD3 and SAC. Episode length is swept from 25 to 300 steps in increments of 25 steps, while the step length scaler is retained constant at 1mm value.

C.3.2.1 Experimental Results

The episode length impacts both training time and inference performance. Here we seek to identify the optimal episode length to simultaneously satisfy both these criteria. Figure C.8 illustrates the plot of mean return per step as a function of episode length for continuous action policies trained with TD3 and SAC. A logarithmic curve is fit to the data points yielding an R^2 coefficient of 0.84 for TD3 and 0.87 for SAC. The result affirms our expectation that such a logarithmic relation exists for our environment. Based on this experiment, we concluded that an episode length between 150 and 250 would be optimal

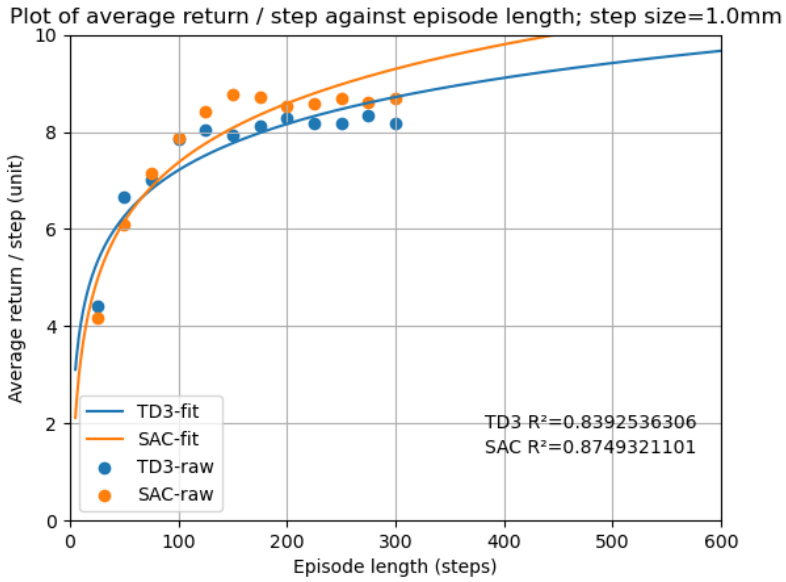


Figure C.8: Relationship between return-per-step and episode length for a continuous action space.

and settled in the middle at 200 steps.

C.4 Multi-Component Iterative Placement

We investigate two important aspects of the multi-component setup. First, we study the impact expert knowledge has on training performance. Second, we look at the effect of the replay buffer on training performance where we consider buffers of fixed size but also resizable ones with a variety of resizing strategies.

C.4.1 Impact of Expert Knowledge on Learning Performance

Equation 4.10 describes the normalised reward signal relative to the initial conditions and best-known historical value for EW and HPWL, otherwise referred to as expert knowledge. This section looks at the impact prior expert knowledge has on both training and performance. Two experimental setups are configured that on the onset of training, respectively start with no expert knowledge and with expert knowledge identified by prior agents. In the case of the former, the agents have to identify the best wirelength values as part of the exploration process. In contrast, the latter requires practically no change to the expert parameters because they are likely very close to the optimal values already.

Configuration	Expert	TD3	TD3 (Pruned)	SAC	% SAC
EW=6, HPWL=2, Overlap=2	N	546.37 ± 302.58	745.13 ± 329.44	748.95 ± 259.48	0.51%
EW=2, HPWL=6, Overlap=2	N	706.36 ± 356.60	706.36 ± 356.60	773.48 ± 264.67	9.50%
EW=2, HPWL=2, Overlap=6	N	623.77 ± 274.08	623.77 ± 274.08	672.39 ± 194.23	7.79%
EW=4, HPWL=4, Overlap=2	N	670.13 ± 322.51	670.13 ± 322.51	769.29 ± 247.17	14.80%
EW=6, HPWL=2, Overlap=2	Y	503.08 ± 342.50	718.01 ± 344.83	759.07 ± 239.56	5.72%
EW=2, HPWL=6, Overlap=2	Y	724.78 ± 306.57	724.78 ± 306.57	769.66 ± 241.95	6.19%
EW=2, HPWL=2, Overlap=6	Y	645.23 ± 256.83	645.23 ± 256.83	692.76 ± 192.30	7.37%
EW=4, HPWL=4, Overlap=2	Y	312.74 ± 296.15	722.33 ± 319.39	742.07 ± 225.10	2.73%

Table C.4: Average return for experiments studying the impact of expert knowledge on learning performance.

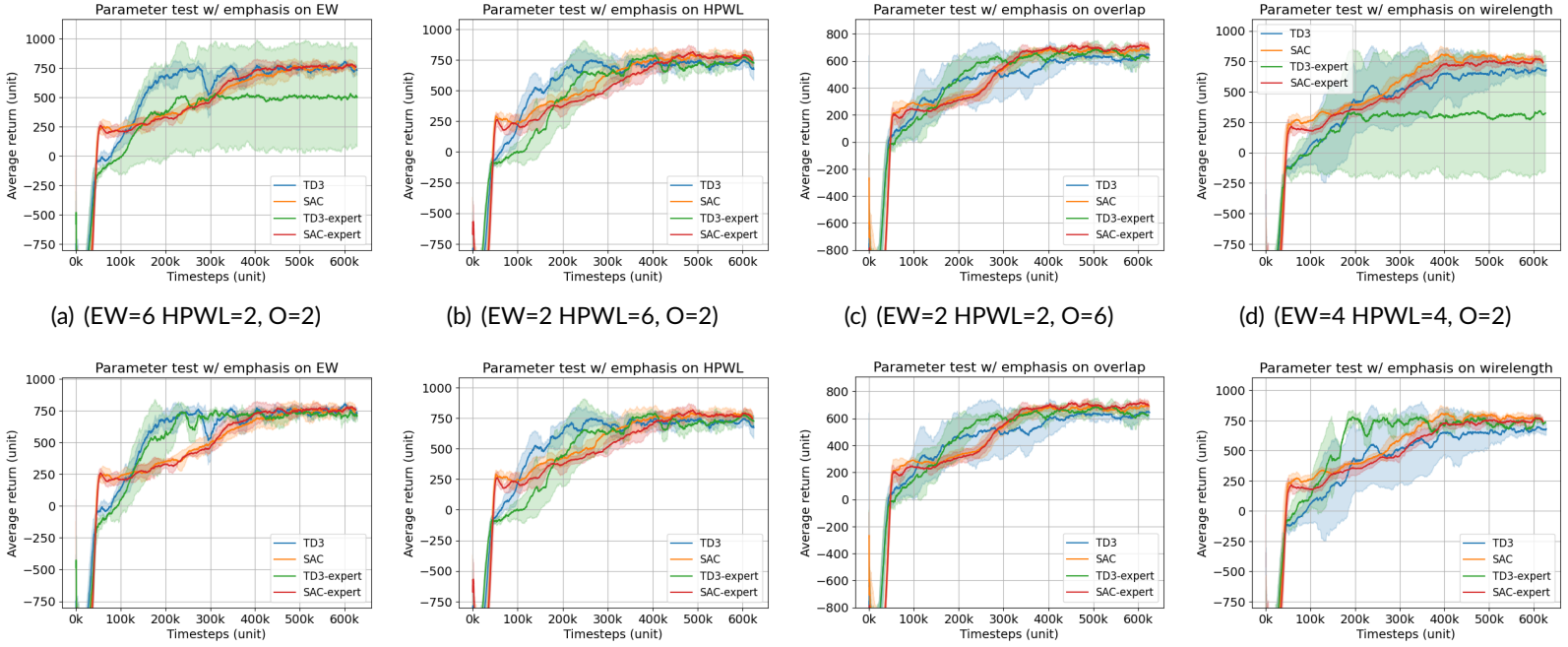
C.4.1.1 Experimental Results

Figure C.9 presents return charts for the experiment detailed previously. Recall that multi-agent experiments are averaged over four runs, and Sub Figures C.9(a) to C.9(d) illustrate this aggregation. Occasionally, TD3 fails to learn a policy. The return plots manifest with a substantially lower average accompanied by a significant standard deviation as demonstrated by Sub Figures C.9(a) and C.9(d). Sub Figures C.9(e) to C.9(h) prune the failed runs and compute the average based on the available runs. Table C.4 summarises the experimental results, with the percentage difference computed between SAC and TD3 with the failed runs removed. SAC outperforms TD3 on all runs and on average by 8.15% and 5.5% with and without expert knowledge respectively. Additionally, SAC is more stable throughout training, evidenced by two factors: exhibits a lower standard deviation across runs, and we did not witness a single scenario where SAC failed to learn a policy.

Table C.5 shows the return difference when using known values for expert parameters. On average, TD3 leads to a slightly increased return of 2.52% while SAC demonstrates negligible improvement. Furthermore, when expert data was used, (3 / (4*4) = 18.75%) TD3 runs failed to learn a policy, in contrast with (1 / (4*4) = 6.25%) when leaving it up to the agent, to identify the expert values. Starting with known best historical parameters may create a more challenging situation where the agent needs to maximize the reward.

Configuration	% Expert TD3	% Expert SAC
EW=6, HPWL=2, Overlap=2	-3.78%	1.35%
EW=2, HPWL=6, Overlap=2	2.61%	-0.50%
EW=2, HPWL=2, Overlap=6	3.44%	3.03%
EW=4, HPWL=4, Overlap=2	7.79%	-3.67%
Mean	2.52%	0.05%

Table C.5: Difference in average return due to expert knowledge.



(a) (EW=6 HPWL=2, O=2) (b) (EW=2 HPWL=6, O=2) (c) (EW=2 HPWL=2, O=6) (d) (EW=4 HPWL=4, O=2)

(e) (EW=6 HPWL=2, O=2) (f) (EW=2 HPWL=6, O=2) (g) (EW=2 HPWL=2, O=6) (h) (EW=4 HPWL=4, O=2)

Figure C.9: Average return illustrating the impact of expert knowledge on the training process. Each result is aggregated over a maximum of four runs when applying pruning. The top four Figures aggregate all four training runs while the lower four prune the ones that did not manage to learn a policy.

C.4.2 Impact of Replay Buffer on Adaptive Reward Learning

The replay buffer plays a crucial role in the multi-agent setup. As mentioned in the previous section, the agent will likely find better values for the expert parameters as part of the trial-and-error search process. Furthermore, we rarely expect it to be the case where the agent has access to the best values at the onset of training. For this reason, the data in the replay buffer will be relatively inconsistent as the agent's policy becomes better and identifies improved expert parameters. Table C.6 summarises the experimental setup that tests the impact of the replay buffer on learning capacity. First, we investigate a fixed replay buffer of various sizes. Secondly, we test a variable-size replay buffer subject to a resizing strategy. The resizing strategy updates the size of the replay buffer after carrying out several steps equal to the latest buffer size. The overarching idea is that initially, the policy is terrible and expert parameters are poor. Therefore we want to discard irrelevant data points as quickly as possible. In later stages, when the policy matures and expert values are close to their optimal for a given layout, the replay buffer is large and can thus sample mixed batches of consistent data points.

Replay buffer size	**Variable**	**Resizing strategy**
3e5	No	-
6e5	No	-
1.2e6	No	-
1.8e6	No	-
3e6	No	-
25e3	Yes	Double every <replay buffer size>steps
25e3	Yes	Tripe every <replay buffer size>steps
25e3	Yes	Quadruple every <replay buffer size>steps

Table C.6: Fixed and variable-size replay buffer configurations for investigating the impact of an adaptive reward signal on learning performance.

C.4.2.1 Experimental Results

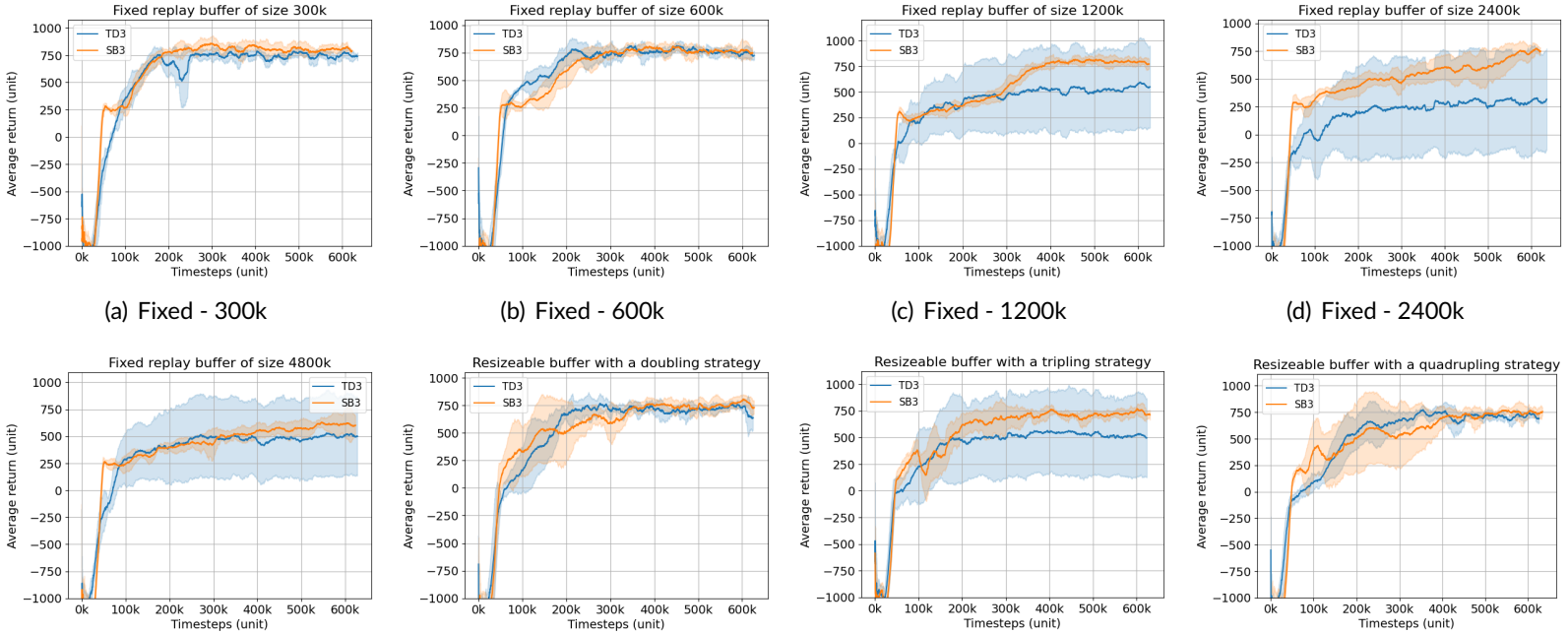
Figure C.10 illustrates the raw return plots for the configurations presented in Table C.6, aggregated over four runs. Figure C.11 illustrates the same results after pruning the failed runs. Concerning SAC, the average return and associated standard deviation are inversely and directly proportional to the buffer size. This relationship, while seemingly present, is not as clear for TD3 due to policies that fail to learn. For instance, the setup with a buffer size of 2400k has a 50% success rate and, compared to other fixed configurations, is an outlier. However, according to Table B.5 the average number of components per layout

in the training set is five, which means that after 600k steps, the number of data points will be roughly 3000k. Therefore for large replay buffers, the degradation in performance is likely to be caused by sample inconsistencies due to the adaptive reward signal.

A variable replay buffer was outperformed in all scenarios with both average return and standard deviation metrics. Both SAC and TD3 favoured a small replay buffer of 1200k or less. Under this condition, SAC attained the best return with 1200k, which was also the better choice for TD3 due to a comparable return and a standard deviation that is 4.1% lower than average. In conclusion, we expect better results for the multi-agent PCB placer if the variable replay buffer is replaced with one having a fixed size of 1200k.

Strategy		**TD3**	**TD3 (Pruned)**	**SAC**	**% SAC**
Fixed	300k	741.9 ± 351.83	741.9 ± 351.83	797.05 ± 312.65	7.43%
Fixed	600k	750.30 ± 345.75	750.30 ± 345.75	760.72 ± 267.88	1.39%
Fixed	1200k	544.19 ± 292.91	744.94 ± 311.21	788.27 ± 267.30	5.82%
Fixed	2400k	296.96 ± 346.73	690.96 ± 349.07	714.09 ± 246.71	3.35%
Fixed	4800k	500.81 ± 251.01	685.28 ± 262.10	641.19 ± 231.45	-6.88%
Variable	Double	717.13 ± 368.46	717.13 ± 368.46	749.97 ± 283.57	4.58%
Variable	Triple	510.17 ± 318.28	699.09 ± 339.15	723.89 ± 291.74	3.55%
Variable	Quadruple	707.59 ± 328.73	707.59 ± 328.73	750.28 ± 288.99	6.03%

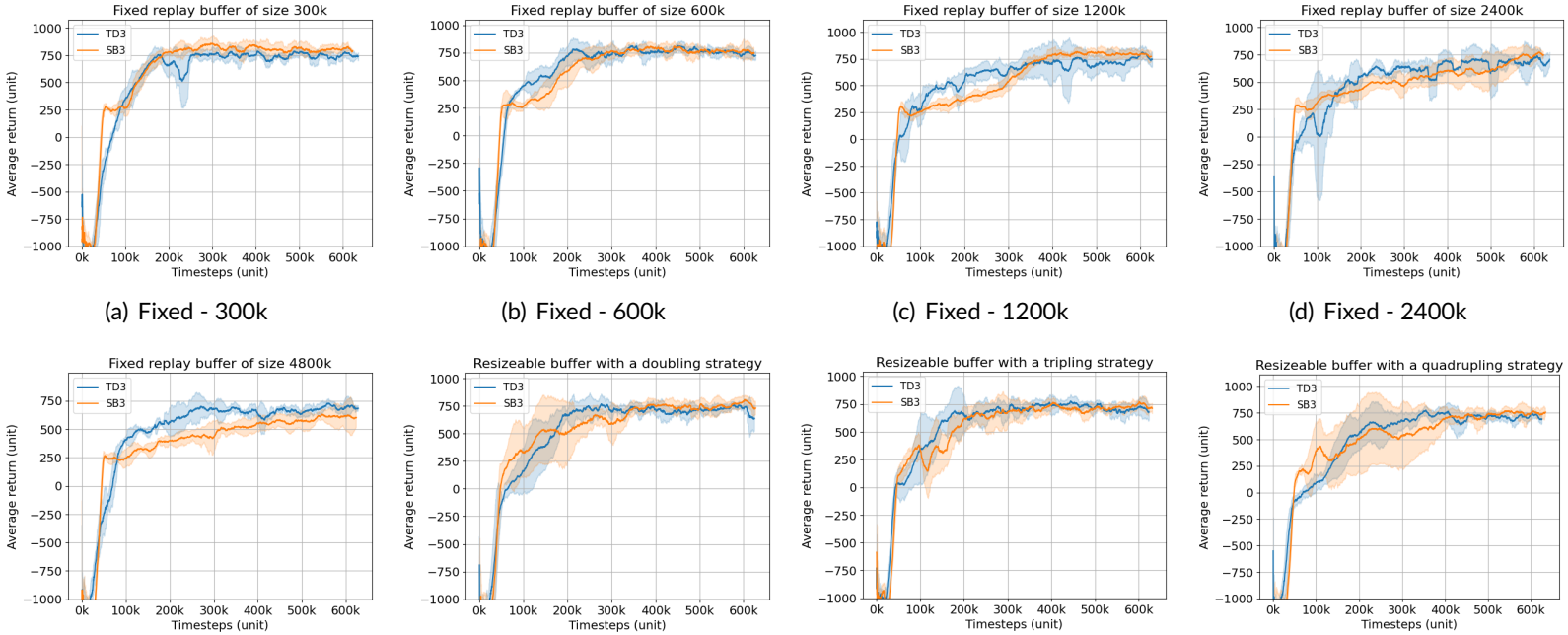
Table C.7: Average return for replay buffer experiments investigating the impact of an adaptive reward signal on learning performance.



(a) Fixed - 300k (b) Fixed - 600k (c) Fixed - 1200k (d) Fixed - 2400k

(e) Fixed - 4800k (f) Variable - double (g) Variable - triple (h) Variable - quadruple

Figure C.10: Average return illustrating the impact of the replay buffer size and resizing strategy on the learning process. Each result is aggregated over four runs regardless of learning success. Starting at the top left, the first five charts use a fixed-size replay buffer and the remaining three start with a buffer size of 25k and employ a resizing strategy. Reward signal configuration used is (EW=2 HPWL=6, Overlap=2).



(a) Fixed - 300k (b) Fixed - 600k (c) Fixed - 1200k (d) Fixed - 2400k

(e) Fixed - 4800k (f) Variable - double (g) Variable - triple (h) Variable - quadruple

Figure C.11: Average return illustrating the impact of the replay buffer size and resizing strategy on the learning process (pruned). Each result is aggregated over a maximum of four runs after pruning unsuccessful learning trials. Starting at the top left, the first five charts use a fixed-size replay buffer and the remaining three start with a buffer size of 25k and employ a resizing strategy. Reward signal configuration used is (EW=2 HPWL=6, Overlap=2).

D Additional Background

D.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are parameterisable functions capable of approximating any mathematical relationship of an arbitrary number of variables. They are distinguished from other techniques by the learning process that optimises the parameters with respect to an error signal. This section presents the perceptron model, multi-layer perceptrons and standard training methods.

D.1.1 Perceptron Model

Rosenblatt (1958) proposed the perceptron model depicted in Figure D.1, that comprises a set of weights and a bias analogous to the synapses in the mammalian brain. These are followed by a summation operation and activation function analogous to the neuron's nucleus. In Rosenblatt's model, a binary activation function was used. In a forward pass, the sum of the input data multiplied by the weights together with the bias is computed. If the sum is greater or equal to zero, the neuron will output the binary value of '1' and is said to fire; otherwise will output 0.

$$\sigma(z) = \begin{cases} 1 \; if \; \sum_{n=0}^{n-1} x_i w_i + b \geq 0 \\ 0 \; otherwise \end{cases} \tag{D.1a}$$

$$\sigma(z) = \begin{cases} 1 \; if \; \mathbf{W^T X} + b \geq 0 \\ 0 \; otherwise \end{cases} \tag{D.1b}$$

Equations D.1a and D.1b mathematically describe this operation in an interative multipy-accumulate and vector forms respectively. x_i and $\mathbf{X}$ represent the input data individually and in vector form respectively, w_i and $\mathbf{W}$ are the weights represented separately, and in

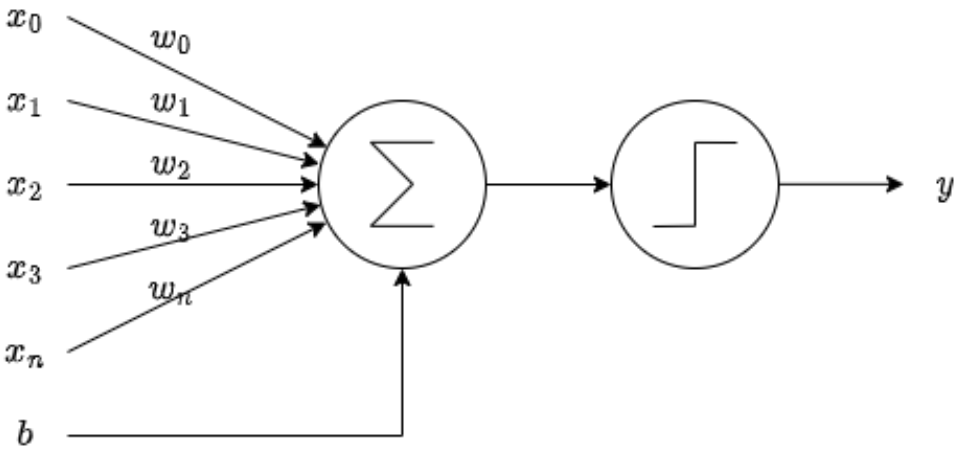


Figure D.1: Rosenblatt's perceptron model (Rosenblatt, 1958).

vector format, and b is the bias term. $\sigma(z)$ is the result of the multiply-accumulate operation between the inputs and the weights together with the bias, which is then clamped to either 0 or 1 by the activation function, yielding the boolean result y. Rosenblatt's neuron model is the fundamental building block of AI systems, with a minor difference. In the mainstream case where gradient-based learning is employed, the discontinuous step activation is replaced with a differential one such as sigmoid or tanh.

D.1.2 Multi-Layer Perceptrons

A MLP constitutes a subset of neural networks. It comprises layers of perceptrons, having an input layer, an output layer and an optional amount of hidden layers. The shape of the input data or problem features restricts the size of the input layer, while the task being carried out defines the size of the output layer. Figure D.2 depicts the architecture of a multi-layer perceptron with one hidden layer for a total of three layers. In a typical forward pass, the input vector is multiplied by the weights of the first layer, and each will sum its inputs with the bias and propagate it through the activation function. The result is input to the next layer and repeated until the output layer is reached.

MLPs are universal function approximators that could approximate any continuous function (Hanin, 2019; Hornik et al., 1989). The quality of the estimated function is subject to the layer width in neurons of the hidden layer. Practically, neural networks with a wide single hidden layer are not very common. Instead, multi-layer deep neural networks are used because they are easier to optimise. The configuration of the hidden neuron has been extensively studied in terms of width and depth Lu et al. (2017).

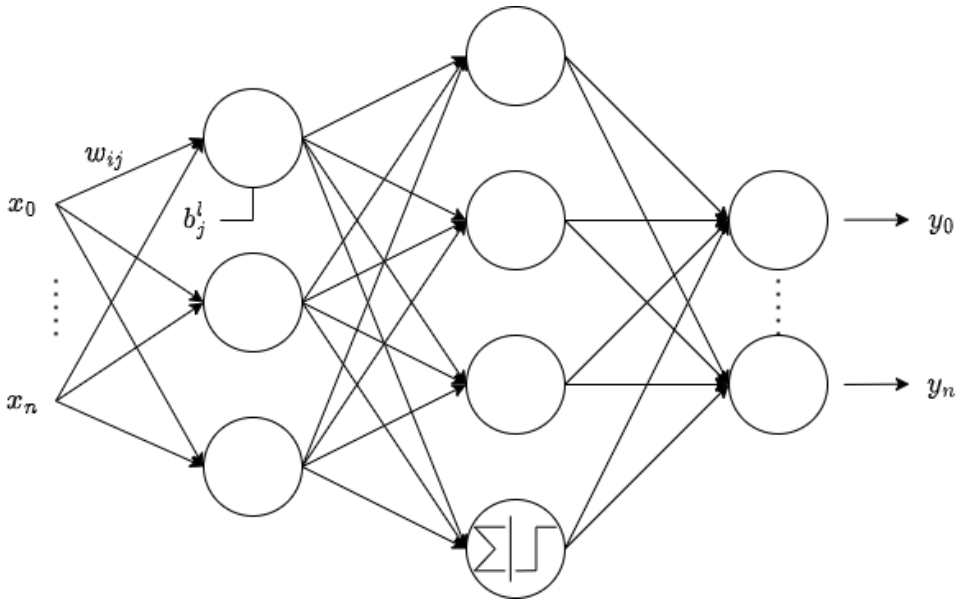


Figure D.2: Architecture of a Multi Layer Perceptron (MLP).

$$\begin{aligned} h_j^l &= \phi(\sum_i x_i w_{ij}^l + b_j^l) \ \forall i, j, l \geq 0 \\ &= \phi(\mathbf{W_j^T X} + b_j^l) \ \forall i, j, l \geq 0 \end{aligned} \tag{D.2a}$$

$$\mathbf{H^l} = \phi(\mathbf{W^T X^l} + \mathbf{B^l}) \ \forall l \geq 0 \tag{D.2b}$$

Equations D.2a and D.2b mathematically capture the propogation of an input through a single layer. h_j^l is the output of the j^{th} neuron in the l^{th} layer and $\mathbf{H^l}$ is the output of all neurons in the l^{th} layer represented in vector format. ϕ is the activation function adopted by each neuron applied to the multiply-accumulate operation between inputs x_i or $\mathbf{X}$, weights w_i or $\mathbf{W}$, together with the bias b_i or $\mathbf{B}$.

D.1.3 Neural Network Training

Multi-layer perceptrons are trained through an optimisation process driven by a cost function that captures the model's error. The cost function collapses the complexity of a neural network down to a single scalar value that may be used to rank and compare solutions. For a fixed topology MLP, the training process adjusts the weights and biases to minimise a loss function. Cross entropy and mean squares error are loss functions that compute the error between the neural network's prediction and a target value. Cross entropy log loss measures the performance of a model whose output is a probability distribution and

is generally used on classification problems. In contrast, Mean Square Error (MSE) loss is used on regression tasks.

Two widespread methods exist for training neural networks, gradient-based learning and black-box optimisation. The backpropagation algorithm (Rumelhart et al., 1986) is frequently used to compute the error gradient with respect to the network's parameters. This approach is effective because by following the gradient, it is guaranteed to find a minimum within the search space. The challenge arises when the search space is non-convex and has many local minima and a single global optimum because the algorithm can converge sub-optimally. Momentum and adaptive learning rate (Kingma and Ba, 2014) are techniques used to move out of local minima in search of better convergence points.

On the other of the spectrum are black-box optimisation methods, which use heuristic algorithms to optimise the parameters of the neural network. Population-based heuristics such as evolutionary algorithms can explore the search space more effectively because multiple solution points are scattered all over the search space. Due to their highly parallel approach, these approaches consume a notable amount of computational power.

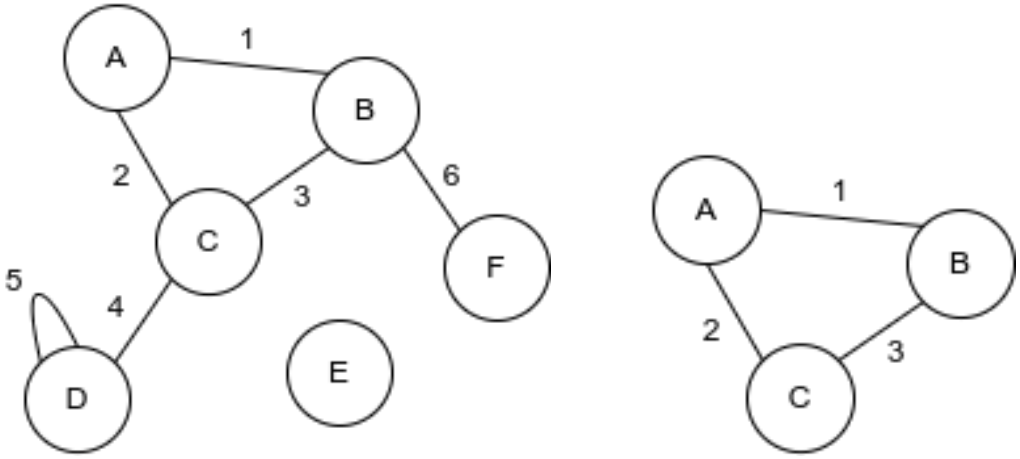
D.2 Graph Theory

Figure D.3 illustrates an undirected graph of six nodes and six edges. Of particular interest is edge five, which originates and terminates at the same node and is called a self-loop.

The order of a graph is equal to the number of nodes, and the size is equal to the number of edges. The degree of a vertex is equal to the number of edges linking to adjacent vertices. An isolated vertex is a particular case that has no adjacent nodes. Concerning Figure D.3(a), the order is six, and the degree of vertices C and F, respectively, are three and one. G is an isolated vertex. Graphs can be classified further and the following list presents some elementary properties.

1. A null graph contains a non-empty set of vertices and an empty set of edges.
2. A regular graph comprises a set of nodes with the same degree. A k-regular graph has all its constituent nodes of degree k.
3. A complete graph has every node connected to every other node in the graph's vertices. An $n-1$-regular graph of order n, where n is the number of nodes, is a complete graph denoted as K_n.
4. The size of a fully connected graph having n nodes is equal to $\frac{n(n-1)}{2}$.

5. A smaller graph derived from a larger graph is called a subgraph and contains a subset of nodes and edges of the original graph. Figure D.3(b) illustrates a subgraph derived from Figure D.3(a). Coincidentally it is also a complete graph.



(a) A Graph (b) Fully connected sub-graph

Figure D.3: Illustration of basic graph structures. On the left is an arbitrary graph, and the one on the right is a derived, fully connected sub-graph.

D.2.1 Mathematical Representation

A graph may be represented using an adjacency list or an adjacency matrix. An adjacency list contains a collection of unordered lists representing connections between graph vertices. In the simple case, a single unordered list within the collection is a pair that represents a connection between two vertices. The adjacency list for the graph in Figure D.3(a) is shown in Equation D.4b. For completeness, the set of nodes is also listed in Equation D.4a. However, it is not limited to a pair and can encapsulate an arbitrary number of vertices, resulting in a hyperedge.

A finite graph may also be represented with a square matrix of booleans, where a logic one indicates the presence of an edge and a logic zero its absence. The adjacency matrix corresponding to the graph in Figure D.3(a) is listed in Equation D.3.

$$\mathbf{A_m} = \begin{array}{c} \begin{matrix} \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \\ \text{F} \end{matrix} \tag{D.3}$$

$$V = \{A, B, C, D, E, F\} \tag{D.4a}$$

$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}, \{D, D\}, \{B, F\}\} \tag{D.4b}$$

D.3 Net Models

A net consists of an arbitrary number of pins, each having a position (x_i, y_i). A fundamental property of a net is that all pins must have the same electric potential. In other words, they must be connected by the same wire. A net can be represented using a graph where nodes represent pins and edges indicate connections between pins. A hyperedge may also be used linking multiple nodes. A net model describes how to measure or estimate the length of a net. This section presents some standard net models used to approximate the length of a net. These are typically categorised based on differentiability.

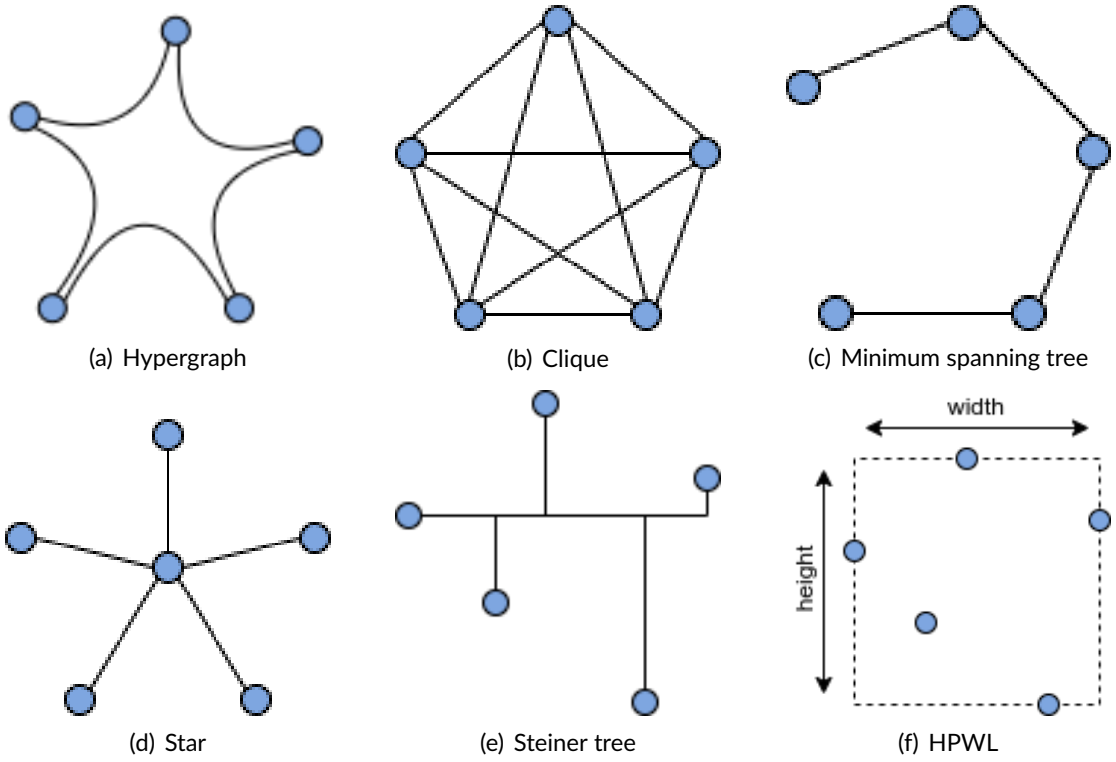
The clique net model uses all two-pin connections of a net resulting in an amount equal to the size of a regular graph. The minimum spanning tree utilises a minimal amount of edges equal to N-1 and attempts to identify the shortest path that connects all constituent pins. The downside is that its construction has a time complexity larger than O(N). A star net model has a runtime complexity of O(N) and uses a virtual node at the centre of the net creating N two-pin connections are between the pins and the virtual node.

A Steiner tree adds additional pins to create a path strictly composed of vertical and horizontal two-pin connections. A minimal Steiner tree results in a set of edges that sum up to a minimal net length. However, finding the optimal Steiner is an NP-hard problem, albeit near-optimal approximations are available.

Half Perimeter Wire Length (HPWL) constructs a minimum bounding rectangle enclosing all pins, making a net. As the name implies, the wirelength is approximated using half the perimeter of the rectangle.

D.3.1 Differentiable Net Models

Differentiable net models mathematically approximate the wirelength of a net while being able to compute its derivate analytically. Often this is required by analytic placers as a replacement for HPWL, which is not a differentiable net model. The quadratic length model



(a) Hypergraph (b) Clique (c) Minimum spanning tree

(d) Star (e) Steiner tree (f) HPWL

Figure D.4: Illustration of different wirelength models.

sums half the euclidean length of every two-pin connection in a net and is employed by quadratic placers. Multi-pin nets are often modelled using a clique or star models to decompose into two-pin connections. Two common differentiable wirelength models often used by nonlinear analytic placers are the lp-norm and LSE. Chan et al. (2005) compares the performance of the two and conclude that, on average, the placers using LSE outperform those modelling wirelength with lp-norm.

The quadratic length model sums half the euclidean length of every two-pin connection in a net. Equation D.5 contains the quadratic wirelength cost of a netlist. Half the quadratic length is considered to simplify the derivative. Multi-pin nets are often modelled using a clique or star models to decompose into two-pin connections.

$$\sum_{e \in E} \frac{1}{2} \left(\sum_{v_i, v_j \in e} w_{x,ij} (x_i - x_j)^2 + \sum_{v_i, v_j \in e} w_{y,ij} (y_i - y_j)^2 \right) \tag{D.5}$$

The LSE approximation and the Lp-norm are sometimes used to more accurately approximate smooth HPWL. Equation D.6a illustrates the LSE where γ is used to control the accuracy of the approximation. As it approaches zero, the LSE cost approaches that of HPWL as shown in Equation D.6b. Similarly, Equation D.7a shows lp-norm with the

difference that converges to the HPWL as p approaches infinity, also noted by Equation D.7b. In both cases, finite numerical precision of IEEE754 floating point arithmetic has to be considered to prevent arithmetic overflows during implementations. Chan et al. (2005) compare the performance of the two and conclude that, on average, the LSE outperforms the lp-norm.

$$LSE_e = \gamma \left(log \sum_{v_k \in e} e^{\frac{x_k}{\gamma}} + log \sum_{v_k \in e} e^{-\frac{x_k}{\gamma}} + log \sum_{v_k \in e} e^{\frac{y_k}{\gamma}} + log \sum_{v_k \in e} e^{-\frac{y_k}{\gamma}} \right) \tag{D.6a}$$

$$\lim_{\gamma \to 0} LSE_e = HPWL_e \tag{D.6b}$$

$$Lpnorm_e = \left(\sum_{v_k \in e} x_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} x_k^{-p} \right)^{-\frac{1}{p}} + \left(\sum_{v_k \in e} y_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} y_k^{-p} \right)^{-\frac{1}{p}} \tag{D.7a}$$

$$\lim_{p \to \infty} Lpnorm_e = HPWL_e \tag{D.7b}$$

E Machine Information and Statistics

This section presents machine information and detailed experiment statistics. First, comprehensive details are presented on the machine being used, including hardware and software specifications. Secondly, a detailed breakdown of the experiment run time for each section of the thesis is provided to understand the experiment's results better.

E.1 Machine Information

The experiments described in this thesis were performed on a Ubuntu 18.04 Linux machine whose details are tabulated in Table E.1.

Component	Value
CPU	Intel Core i7-10700K @ 3.8GHz 8C/16T
Memory	64GB
GPU	Nvidia GeFore GTX 1080 8GB
GPU Driver	470.141.03 CUDA 10.2
Operating System	Ubuntu #147 18.04.1
Kernel Release	5.4.0-131

Table E.1: Machine specifications

E.2 Experiment Runtimes

Statistics relating to training time and the number of interaction steps carried out in this thesis, excluding supplementary experiments, are summarised in this Section. Average durations were derived from the runtimes logged in Tensorboard (Abadi et al., 2015). The experiments were parallelised with a factor of six to eight, and while we report the running time per experiment, the single-threaded performance is expected to be higher. Table E.2

summarises the experiment run time for the constructive placement methodology. Most of the time is consumed by the hyperparameter and NAS.

Experiment	Algorithm	Configurations	Runs / Configuration	Time / Run (hr)	Epochs	Training Steps	Total Experiment Time (hr)	Total Experiment Steps (millions)
Wirelength Predictor NAS - GCN	-	128	4	0.15	100	-	76.8	-
Wirelength Predictor NAS - GraphConv	-	128	4	0.15	100	-	76.8	-
Wirelength Predictor NAS - GAT	-	128	4	0.15	100	-	76.8	-
Constructive Placement	TRPO	1	4	5	-	5E6	20	20
Constructive Placement	PPO	1	4	4	-	5E6	16	20

Table E.2: Constructive Placement Experiment statistics

Table E.3 summarises the run times of experiments conducted for the single-component iterative placement methodology. Preliminary experiments with reward signal R_1 took up little time. By contrast, significant computation time was expended on R_2 and R_3 studying various parameters, including action spaces, hyperparameter optimisation and NAS, reward signal variations and their impact on generalisation performance. These efforts were compounded due to the consideration of two policy optimisation methods (TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017)) and two actor-critic methods (TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018)).

Experiment	Algorithm	Configurations	Runs / Configuration	Time / Run (hr)	Training Steps	Total Experiment Time (hr)	Total Experiment Steps (millions)
Discrete R_1 - Individual	TRPO	4	6	1.5	1E6	24	36
Discrete R_1 - Individual	PPO	4	6	1	1E6	24	24
Discrete NAS	TRPO	64	1	1.5	1E6	64	96
Discrete NAS	PPO	64	1	1	1E6	64	64
Discrete R_2 - Individual	TRPO	12	6	1.5	1E6	72	108
Discrete R_2 - Multiple	TRPO	4	6	3	2E6	48	72
Discrete R_2 - Individual	PPO	12	6	1	1E6	72	72
Discrete R_2 - Multiple	PPO	4	6	2	2E6	48	48
Continuous NAS	TRPO	64	2	1.5	1E6	128	128
Continuous NAS	PPO	64	2	1	1E6	128	128
Continuous NAS	TD3	64	2	3	1E6	128	384
Continuous NAS	SAC	64	2	2	1E6	128	256
Continuous R_2 - Individual	TD3	16	6	3	1E6	96	288
Continuous R_2 - Multiple	TD3	4	6	6	2E6	48	144
Continuous R_2 - Individual	SAC	16	6	2	1E6	96	192
Continuous R_2 - Multiple	SAC	4	6	4	2E6	48	96
Continuous R_3 - Replay Buffer	TD3	6	6	3	1E6	36	108
Continuous R_3 - Replay Buffer	SAC	6	6	2	1E6	36	72
Continuous R_3 - Individual	TD3	6	6	3	1E6	36	108
Continuous R_3 - Multiple	TD3	2	6	6	2E6	36	72
Continuous R_3 - Individual	SAC	6	6	2	1E6	24	72
Continuous R_3 - Multiple	SAC	2	6	4	2E6	24	48
Total						2616	1408

Table E.3: Single-Component Placement Experiment statistics

Table E.4 summarises the running time of experiments used in the multi-component iterative placement methodology. The runtime increases significantly, in particular, due to

the image processing tasks involved in feature extraction, which considers the interaction of all movable circuit components in an episode step. Furthermore, our vanilla implementations of TD3 and SAC may not be as optimised as those offered out of the box by StableBaselines3 (Raffin et al., 2021), potentially contributing to increased runtime.

Experiment	Algorithm	Configurations	Runs / Configuration	Time / Run (hr)	Training Steps	Total Experiment Time (hr)	Total Experiment Steps (millions)
Parameter Experiments	TD3	16	4	9	6E5	576	192
Parameter Experiments	SAC	16	4	11	6E5	704	192
Ablation Experiments	TD3	16	4	9	6E5	576	192
Ablation Experiments	SAC	16	4	11	6E5	704	192
Total						2560	768

Table E.4: Multi-Component Placement Experiment statistics