Quantum Machine Intelligence (2026) 8:15
https://doi.org/10.1007/s42484-026-00341-4

RESEARCH ARTICLE

Minor embedding for quantum annealing with reinforcement learning

Riccardo Nembrini[1] · **Maurizio Ferrari Dacrema**[1] · **Paolo Cremonesi**[1]

Received: 16 July 2025 / Accepted: 7 January 2026

Abstract
Quantum Annealing (QA) is a quantum computing paradigm for solving combinatorial optimization problems formulated as Quadratic Unconstrained Binary Optimization (QUBO) problems. An essential step in QA is minor embedding, which maps the problem graph onto the sparse topology of the quantum processor and then adjusts the problem weights. The process of mapping the problem variables to the hardware is computationally expensive and scales poorly with increasing problem size and hardware complexity. Existing heuristics are often developed for specific problem graphs or hardware topologies and are difficult to generalize. To address this limitation, we explore the use of machine learning methods, which would allow a much greater degree of flexibility, in particular of Reinforcement Learning (RL). RL offers a promising alternative by treating minor embedding as a sequential decision-making problem, where an agent learns to construct minor embeddings by iteratively mapping the problem variables to the hardware qubits. We propose a RL-based approach to minor embedding using a Proximal Policy Optimization agent, testing its ability to embed both fully connected and randomly generated problem graphs on two hardware topologies, Chimera and Zephyr. The results show that our agent consistently produces valid minor embeddings even when they span over more than a thousand qubits, in particular on the more modern Zephyr topology. Our proposed approach is also able to scale to moderate problem sizes and adapts well to different graph structures, highlighting RL's potential as a flexible and general-purpose framework for minor embedding in QA but also pointing to limitations that will need to be addressed, for example in reducing the number of qubits required.

Keywords Quantum Annealing · Minor Embedding · Reinforcement Learning · Proximal Policy Optimization

1 Introduction

Among the different paradigms of Quantum Computation, Quantum Annealing (QA) operates by representing an optimization problem as an energy minimization one, and then evolving a physical quantum system from an initial default configuration towards a final one that is constructed based on the target problem (Kadowaki and Nishimori 1998; Johnson et al. 2011). QA is often used to solve combinatorial optimization problems formulated as Quadratic Unconstrained Binary Optimization (QUBO) ones. While the QUBO formulation is very general, the physical hardware of a Quantum Processing Unit (QPU), or Quantum Annealer, exhibits a specific topology and connections between the physical qubits. Typical optimization problems in machine learning and other domains are not natively compatible with the hardware connectivity and must be transformed into other equivalent ones having a structure that is compatible with the QPU, in a process called *Minor Embedding* (ME).

Minor embedding requires mapping the problem variables to the hardware and then adjusting the weights associated with the problem graph in order to account for this new topology. The mapping phase, in particular, is an optimization problem of its own and often acts as a computational bottleneck (Ferrari Dacrema et al. 2022), requiring a time that exceeds by orders of magnitude the actual quantum annealing process. Furthermore, existing heuristics lack flexibility because they do not include objective functions that can be adjusted according to the scenario of interest. While their computational cost could be mitigated by pre-computing minor embeddings or developing ad-hoc

✉ Maurizio Ferrari Dacrema
maurizio.ferrari@polimi.it

Riccardo Nembrini
riccardo.nembrini@polimi.it

Paolo Cremonesi
paolo.cremonesi@polimi.it

[1] Politecnico di Milano, Milano, Italy

Published online: 19 February 2026

heuristics for fixed problem graphs, such as fully connected graphs, on specific hardware topologies (Boothby et al. 2016), this strategy is limited by the large number of possible problem graphs and hardware variations due to inactive qubits or differing topological layouts. Moreover, the minor's structure can significantly affect solution quality. For example, longer qubit chains increase the likelihood of errors and inconsistencies during annealing, as well as making it more difficult for the qubits to change their state and therefore optimize the desired objective function. These issues can be mitigated with encoding and decoding schemes to improve robustness (Vinci et al. 2015). The relationship between minor embedding and solution quality, however, also depends on other characteristics of both the problem and the minor embedding itself that go beyond the chain length, including the distribution of the weights (Pellini and Ferrari Dacrema 2024).

These limitations of existing heuristics motivate the search for alternative, more flexible machine learning approaches to minor embedding, that would also allow the freedom to define new objective functions. In this work, we explore the potential of Reinforcement Learning (RL) to build the node mapping required by the minor embedding process by formulating it as a sequential decision-making problem. While RL methods are a recent and relatively underexplored direction, and bring a set of challenges of their own, e.g., long training time and instabilities, they also provide a much higher degree of flexibility and adaptability to changing conditions, which can support their generalization across problem instances and hardware topologies. On account of the limited literature on this task, we design a MLP-based agent architecture aiming to develop a simple, dependable and fast RL-based approach that can be used in practice with limited fine-tuning and serve as a basis for further work.

In summary, the contributions of this work are as follows:

- We approach minor embedding as a sequential decision-making problem and propose a Proximal Policy Optimization (PPO) based agent to generate the mapping between problem variables and hardware qubits required by minor embedding.
- To improve learning efficiency and exploit the inherent symmetries of the hardware topology, we propose a set of data augmentation strategies that enhance generalization and policy robustness in particular on randomly generated problem graphs.
- We conduct a detailed comparison of minor embedding quality, success rate, and qubit efficiency across two widely used quantum hardware topologies, Chimera and Zephyr, highlighting the differences in agent performance under varying connectivity constraints.

The methods, experiments, and results presented in this paper are an extension of our prior work in Nembrini et al. (2024). Our results show that RL agents can produce valid variable-to-qubit mappings for minor embedding, particularly on modern topologies such as Zephyr. The agent is able to generate relatively easily valid minor embeddings that span several hundreds of qubits, sometimes over a thousand, indicating that its modelling is robust and can scale to the large hardware graphs of existing quantum annealers. Taken together, the findings suggest that RL is a promising and flexible framework for addressing minor embedding in QA complementing existing heuristic methods, yet it remains an emerging approach that needs to mature, for example to reduce the number of qubits required. In the discussion we highlight some limitations of the proposed method and suggest possible directions for future developments.

2 Background

This section provides the necessary background on the three key topics: the computational paradigm of Quantum Annealing, the Minor Embedding problem, and the Reinforcement Learning paradigm which our proposed method is based on.

2.1 Quantum annealing

Quantum Annealing (QA) is a metaheuristic quantum algorithm designed to solve combinatorial optimization problems by exploiting the principles of quantum mechanics (Kadowaki and Nishimori 1998; Farhi et al. 2000; Johnson et al. 2011). The approach relies on formulating the optimization problem as an energy minimization of a real physical system, encoding it as a Hamiltonian H_P (known as the problem Hamiltonian) whose ground state corresponds to the optimal solution.

QA operates by initializing the quantum system in the ground state of a simple Hamiltonian H_0, for which the ground state is easy to prepare, typically an equal superposition. Over the course of the annealing schedule, the system evolves according to a time-dependent Hamiltonian of the form:

$$H(t) = (1 - s(t))H_0 + s(t)H_P, \tag{1}$$

where $t \in [0, T]$ is the time parameter, T is the total annealing time, and $s(t)$ is a monotonically increasing function that controls the evolution schedule satisfying $s(0) = 0$ and $s(T) = 1$. If the evolution is sufficiently slow, according to the requirements of the adiabatic theorem (Born and Fock 1928), the system will remain in its instantaneous ground

state. At the end of the evolution, when $H(T) = H_P$, the ground state of the system will also correspond to the ground state of H_P hence to the solution of the problem.

The problem Hamiltonian H_P is typically constructed from a QUBO (Quadratic Unconstrained Binary Optimization) formulation:

$$\min_{x\in\{0,1\}^n} x^\top Q x, \tag{2}$$

where $Q \in \mathbb{R}^{n\times n}$ is a symmetric matrix defining the cost function. This classical problem is mapped onto an Ising Hamiltonian of the form:

$$H_{\text{Ising}} = -\sum_{i=1}^{n} h_i s_i - \sum_{i=1}^{n}\sum_{j=i+1}^{n} J_{ij} s_i s_j, \tag{3}$$

where $s_i \in \{-1, +1\}$ are spin variables representing the state of qubit i, $h_i \in \mathbb{R}$ are linear biases, and $J_{ij} \in \mathbb{R}$ are quadratic couplings between qubits. By looking at Eq. 2 and 3 one can see that there is a biunivocal correspondence between the two. Hence, by constructing a quantum system that implements and minimizes an Ising Hamiltonian, one can also minimize QUBO problems. This approach has gained popularity because it allows to formulate many difficult problems rather easily (Lucas 2014; Glover et al. 2022). Many applications of QA have been proposed in the fields of machine learning (Neven et al. 2009; Mandrá et al. 2016; O'Malley et al. 2017; Mott et al. 2017; Kumar et al. 2018; Ottaviani and Amendola 2018; Neukart et al. 2018a, b; Willsch et al. 2020; Nembrini et al. 2021, 2022; Ferrari Dacrema et al. 2022; Pasin et al. 2024; Carugno et al. 2024), chemistry (Hernandez and Aramon 2017; Xia et al. 2018; Streif et al. 2019; Micheletti et al. 2021), as well as logistics and optimization (Rieffel et al. 2015; Stollenwerk et al. 2017; Ikeda et al. 2019; Ohzeki 2020; Carugno et al. 2022; Chiavassa et al. 2022).

Despite its general formulation, in practice QA is implemented on hardware platforms such as D-Wave quantum processors, where the qubits are laid out in sparsely connected topologies, see Fig. 1. These topologies are typically composed of a repeated grid of small subgraphs called *unit cells*, whose structure depends on the topology. Given that the optimization problem must be physically encoded into the hardware, there must be a biunivocal correspondence between the two. This is obtained via the *Minor Embedding*, in which the optimization problem is first represented as a graph, where problem variables that have a non-zero quadratic coefficient are connected, and then it is transformed in a new and equivalent one which can be directly implemented on the physical hardware.

2.2 Minor embedding

When a problem is expressed in the QUBO formalism, it is typically assumed that all problem variables can interact freely. However, executing such a problem on a quantum annealer requires mapping it to the physical hardware, which imposes strict topological constraints. Each qubit in the quantum processing unit (QPU) can only interact with a limited set of neighbouring qubits, determined by the architecture (e.g., Chimera or Zephyr), see Fig. 1. As a result, the problem must be transformed to conform to the physical connectivity of the device, a process known as *Minor Embedding* (ME).

The central idea of ME is to first represent the optimization problem as a graph, where the variables that are associated with a quadratic coefficient are connected. Then, if a problem variable must be connected to more variables

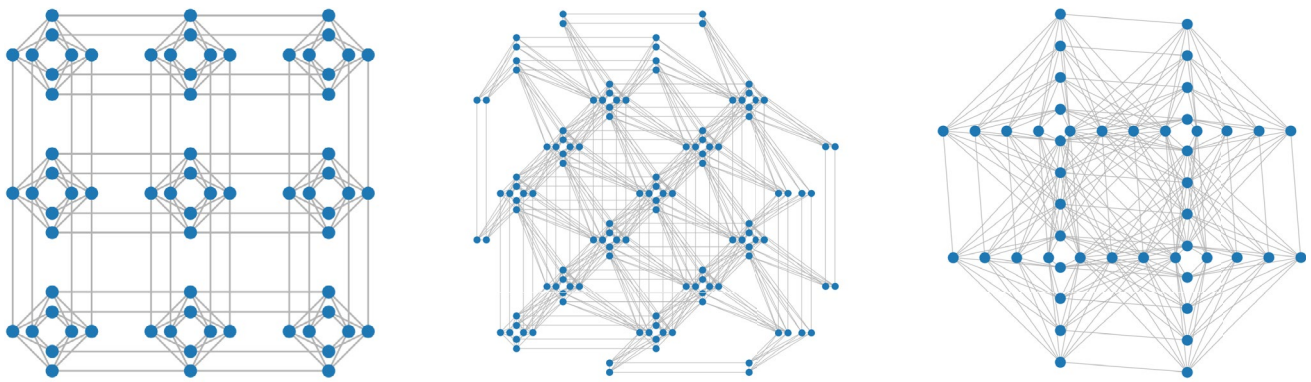


Fig. 1 Portions of the currently existing topologies of physical quantum annealers produced by D-Wave, from oldest (Chimera) to most recent (Zephyr). The main difference between them is the number of connections between qubits, up to 6 in Chimera, up to 15 in Pegasus and up to 20 in Zephyr

than the physical connectivity of the hardware allows, that variable will be represented using multiple physical qubits, i.e., a *chain*, and the connections will be split among them. The chains form connected sub-graphs within the hardware topology and are forced to act coherently by applying strong coupling coefficients (J_{ij}) between their qubits. These couplings encourage all qubits in a chain to be in the same state during the annealing process.

Formally, ME involves two main steps: *node mapping*, where each node of the problem graph G is assigned to one or more physical qubits in the hardware graph H, and *parameter setting*, which distributes the bias (h_i) and coupling (J_{ij}) values among the embedded variables. The node mapping phase is typically the most computationally expensive one, which is why we chose to focus on it. The resulting embedded graph $G_{\text{emb}} \subseteq H$, also called *minor*, contains intra-chain edges (within chains) and inter-chain edges (between different chains). The minor embedding is *valid* if contracting all intra-chain edges in G_{emb} recovers the original problem graph G.

The embedded Ising energy function can be written as:

$$\mathcal{E} = \sum_{i \in V(G)} \left(\sum_{i_k \in V(C_i)} h'_{i_k} s_{i_k} + \sum_{i_p, i_q \in E(C_i)} F_i^{pq} s_{i_p} s_{i_q} \right) + \sum_{i,j \in E(G) \setminus E(C)} J_{ij} s_i s_j, \tag{4}$$

where $V(G)$ is the set of nodes in graph G, $E(G)$ is the set of edges in graph G, C_i denotes the chain assigned to problem variable i, and $F_i^{pq} < 0$ are the intra-chain couplings. The original bias h_i is partitioned among the qubits in the chain such that $\sum_{i_k \in V(C_i)} h'_{i_k} = h_i$, while inter-chain couplings J_{ij} remain unchanged.

Minor embedding quality plays a crucial role in the effectiveness of quantum annealing. Long chains are more prone to *chain breaks*, which occur when qubits within a chain do not reach the same final state, meaning that the quantum annealer has moved away from the ground state. Chain breaks can yield suboptimal or unfeasible solutions, where constraints are not met. Therefore, minimizing chain length, maintaining compact minors, and tuning chain strengths appropriately are essential objectives in the minor embedding process.

In light of this, it is clear that the node mapping phase of minor embedding is NP-hard and that exact solutions are computationally intractable in the general case. For this reason various heuristic methods have been proposed (Cai et al. 2014; Boothby et al. 2016; Fang and Warburton 2020; Bernal et al. 2020; Pelofske 2024). One of the most commonly used tools is `minorminer` (Cai et al. 2014)[1], developed by D-Wave, which as we will describe implements a stochastic algorithm to construct a valid minor of a problem graph into a hardware graph like Chimera or Zephyr. However, these methods are often developed for specific hardware or graph topologies and, being heuristics, do not allow direct control over the actual optimization objective.

2.3 `Minorminer`

One of the most commonly used tools for ME is `minorminer` (Cai et al. 2014), developed by D-Wave, which implements a stochastic algorithm to construct a minor embedding of a problem graph into a hardware graph like Chimera or Zephyr.

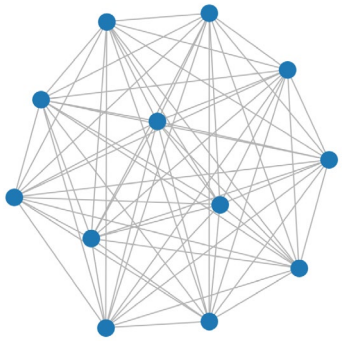
The basic idea is to embed one problem variable at a time. `minorminer` selects a not-yet-embedded problem variable and tries to assign it to a chain of physical qubits that would allow it to reach the other problem variables it must be connected with, according to the problem graph. If a suitable chain is found, the variable is added to the minor; otherwise, the algorithm can backtrack, removing previously assigned chains and trying different paths. To do this, `minorminer` uses a greedy and randomized strategy:

- Problem variables are considered in an order that prioritizes high connectivity or other heuristics.
- For each variable, it attempts to grow a chain using shortest paths to already embedded neighbors.
- If no valid chain is found, it probabilistically selects a different placement or backtracks.

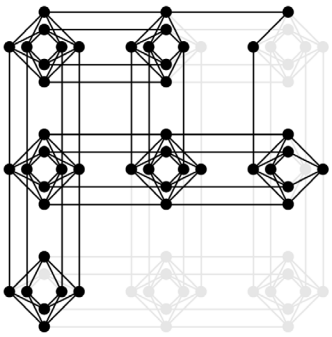
This makes `minorminer` able to escape from local failures. The process continues until all problem variables are embedded, or a timeout or failure condition is reached. The output is a mapping from problem variables to chains of physical qubits. An example of minor embedding of a fully connected graph on different hardware topologies is reported in Fig. 2. `minorminer` does not guarantee optimality, and it is sensitive to the ordering of the problem variables and to random choices during search. For this reason, the minor embedding varies from one run to another. In practice, it is often run multiple times to find better minors, especially for dense problem graphs.

The main strengths of `minorminer` are its ability to handle large and irregular graphs, and its efficiency on modern hardware topologies it is optimized for. However, the time required to find a minor embedding will still be substantial compared to the quantum annealing time, the minor embeddings it finds will still use long chains, depending on the problem, and it does not allow easy customization to tailor its behaviour to specific goals. This motivates the exploration of alternative approaches based on machine learning.

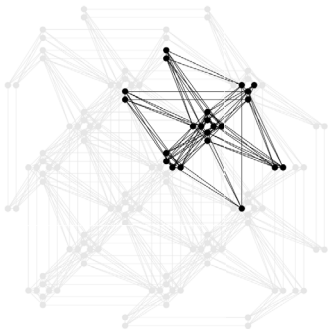
[1] https://github.com/dwavesystems/minorminer



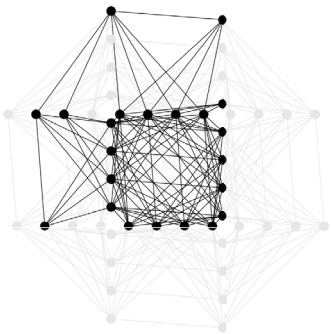
(a) Problem graph G



(b) Chimera (47 nodes)



(c) Pegasus (23 nodes)



(d) Zephyr (22 nodes)

Fig. 2 Example of node mappings from a fully connected graph G of 12 nodes on different QA hardware topologies, obtained with `minorminer`. Highlighted in 2b, 2c and 2d are only the nodes and edges that are part of the minor embeddings (respectively 47, 23 and 22 nodes). The chains are not differentiated for readability

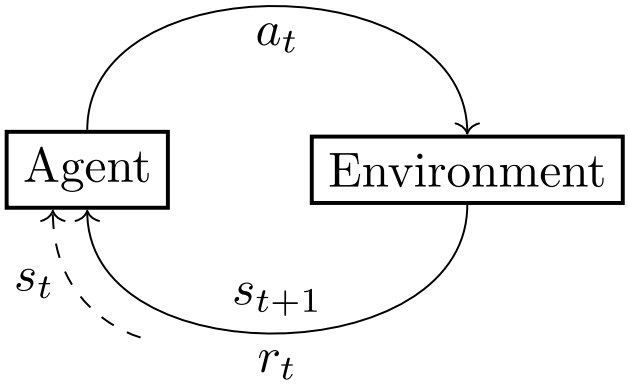


Fig. 3 The fundamental interaction loop of RL. At time step *t* the agent, after observing the environment's state s_t, performs actions a_t. The environment reacts to such action by giving reward r_t to the agent and then transitioning to state s_{t+1}. The new state will be subsequently observed by the agent at the next time step

2.4 Reinforcement learning

Reinforcement Learning (RL) is a machine learning paradigm that involves an agent iteratively interacting with an environment described by a Markov Decision Process (MDP) (Sutton and Barto 2018). At each discrete time step *t*, the agent observes a state s_t of the environment and selects an action a_t based on this observation. The chosen action influences the environment, causing a transition to a subsequent state s_{t+1} according to the dynamics defined by the underlying MDP, and results in receiving a scalar reward signal r_t. This interaction is cyclically repeated, as represented in Fig. 3, until a termination condition is reached at the final step *T*. At that point, the agent has completed an *episode*, collecting a number of rewards. The goal of the agent is to maximize the expected cumulative reward, also known as the *return*, obtained during each episode and defined as $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$, where $\gamma \in [0, 1]$ is a discount factor that balances the importance of immediate versus future rewards. Maximizing this return allows the agent to learn an optimal strategy, called *policy*, for navigating the state-action space. Episodes are thus repeated to allow the agent to systematically explore diverse state-action configurations, thereby enabling learning of effective strategies.

As an illustrative example, consider a robotic agent tasked with traversing a field containing obstacles. In such a scenario, the state observation might include the robot's

current position within the field and the locations of nearby obstacles. Available actions could consist of discrete movements, such as steps in one of the four cardinal directions. Following each action, the agent receives a numerical reward reflecting its effectiveness; for instance, a small negative reward for each move could incentivize efficient paths by penalizing unnecessary steps and a large negative reward could discourage attempting to move in a direction that is obstructed. Upon execution of an action, the environment transitions to a new state corresponding to the updated position of the robot. An episode concludes upon reaching a specified destination or encountering a termination condition (e.g., collision with an obstacle or successful traversal). By repeating episodes under varying conditions, the agent progressively learns improved strategies to navigate the environment effectively.

This framework exemplifies the fundamental reinforcement learning paradigm adopted throughout this work. The training algorithm is introduced in the next sections.

2.4.1 Actor-critic methods

Actor-critic methods constitute a prominent class of reinforcement learning algorithms characterized by employing two complementary structures: the *actor*, which selects actions based on the current policy, and the *critic*, which estimates the value associated with states or state-action pairs (Sutton and Barto 2018; Konda and Tsitsiklis 1999; Mnih et al. 2016). Within this framework, the policy π is formally defined as a probability distribution over possible actions given a state:

$$\pi(a|s) = P(a_t = a \mid s_t = s). \tag{5}$$

The agent aims to find an optimal policy π^*, which maximizes the expected cumulative reward, known as the *return*, defined as:

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}, \tag{6}$$

where $\gamma \in [0, 1]$ is a discount factor balancing immediate versus future rewards, and T denotes the terminal step of the episode.

In actor-critic methods, the actor directly represents the policy, i.e., the learned model that chooses the actions, and is parameterized by a set of parameters θ as $\pi^\theta(a|s)$, while the critic estimates the expected returns associated with states or state-action pairs through a separate parameterized structure, typically denoted by parameters ϕ. Specifically, the critic approximates the *value function*:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \tag{7}$$

which quantifies the expected cumulative reward starting from a given state s and following the current policy π thereafter. Alternatively, the critic may approximate the *state-action value function*:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a], \tag{8}$$

representing the expected cumulative reward obtained by taking action a from state s and subsequently following policy π.

By comparing the state-action value function and the value function, actor-critic methods estimate the *advantage function*:

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi(s), \tag{9}$$

which measures how favourable selecting action a in state s is relative to the average action selection dictated by policy π at that state. Actions yielding positive advantage values are thereby reinforced, whereas actions with negative advantages are discouraged, guiding the policy toward optimal behaviour. Actor-critic methods thus explicitly maintain separate policy and value estimators in order to improve stability, especially in environments characterized by continuous or high-dimensional action spaces (Konda and Tsitsiklis 1999; Grondman et al. 2012).

The subsequent section details the Proximal Policy Optimization (PPO) algorithm, an actor-critic method specifically adopted for training the agents in this work.

2.4.2 Proximal policy optimization

Proximal Policy Optimization (PPO) is an actor-critic algorithm proposed by Schulman et al. (2017), specifically designed to overcome some of the stability and efficiency issues that affected earlier policy gradient methods. PPO aims at improving policy optimization performance while maintaining computational simplicity, achieving a beneficial balance between robustness and ease of implementation.

The PPO algorithm employs two separate neural networks with distinct roles but typically similar structures, see Fig. 4. The first is the *policy network*, i.e., the actor, which parameterizes the policy $\pi^\theta(a|s)$, explicitly defining the probability distribution over actions given state s. The second is the *value network*, parameterized by parameters ϕ, which approximates the value function $v^\phi(s)$, estimating the expected cumulative return starting from a state s under the current policy.

Training in PPO relies on iteratively collecting trajectories of state-action pairs from interactions with the environment,

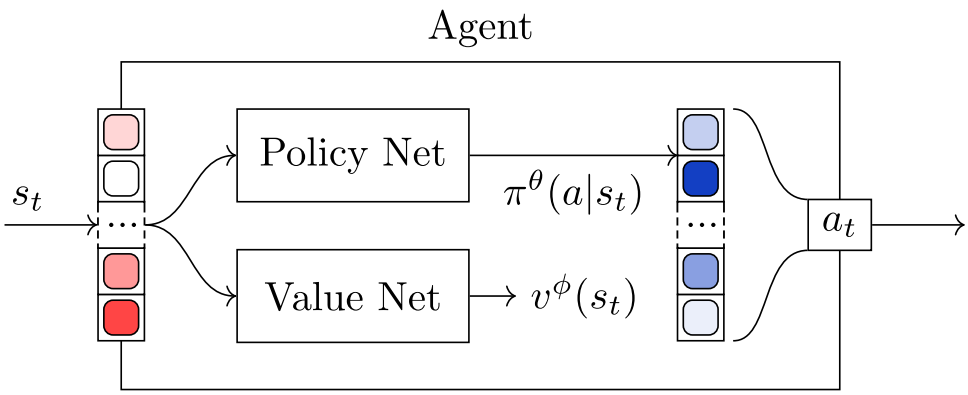


Fig. 4 State s_t is processed by the agent's neural networks. The value network outputs an estimate $v^\phi(s_t)$ of the value function (7), while the policy network outputs a probability distribution $\pi^\theta(a|s_t)$ on the actions (5). Action a_t is sampled from this probability distribution

then performing updates to the network parameters using gradient ascent on an objective function carefully designed to maintain stability. The objective used in PPO is based on a clipped surrogate function, formulated to prevent excessively large policy updates. Specifically, the PPO objective function is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{10}$$

where $\hat{A}_t$ is an estimate of the advantage function (9), computed from the collected trajectories, $r_t(\theta)$ denotes the probability ratio of selecting action a_t under the updated policy relative to the previous policy:

$$r_t(\theta) = \frac{\pi^\theta(a_t|s_t)}{\pi^{\theta_{\text{old}}}(a_t|s_t)}, \tag{11}$$

and ϵ is a hyperparameter controlling the range within which updates to the policy are allowed, ensuring that the policy does not shift excessively during a single update step.

Simultaneously, PPO updates the parameters of the value network by minimizing the squared-error loss between predicted and observed returns, formally expressed as:

$$L_v(\phi) = \mathbb{E}\left[(v^\phi(s_t) - G_t)^2 \right], \tag{12}$$

where ϕ denotes the value network's parameters. Optionally, an additional entropy term may be included in the combined loss function to encourage exploration and prevent premature convergence to suboptimal deterministic policies.

The PPO update procedure typically involves multiple epochs of mini-batch stochastic gradient ascent on the collected trajectory data, thereby improving sample efficiency and further enhancing stability. The choice of hyperparameters, including the clipping threshold ϵ, the discount factor γ, and the learning rate, significantly impacts the performance and convergence behavior of the PPO algorithm.

Due to these desirable properties, PPO has become widely adopted as a powerful baseline for various reinforcement learning applications, ranging from robotic control tasks to complex combinatorial optimization problems, such as the minor embedding scenario addressed in this work (Schulman et al. 2015, 2017).

2.4.3 Invalid action masking

In some cases, based on the state the environment is in, some of the actions may not be available. The set of the available actions can be constrained by Invalid Action Masking (IAM) (Silver et al. 2018; Huang and Ontañón 2022), ensuring the agent is only able to select valid actions. This method consists in masking out invalid actions by setting their probability to zero at the output end of the policy network, thus restricting the set of actions from which the agent chooses.

2.5 Minor embedding and machine learning

While the application of Machine Learning to the minor embedding problem is an emerging area, the literature on this specific task remains limited. One notable contribution that also employs Reinforcement Learning is the *CHARME* framework, presented by Ngo et al. (2025). This method adopts a different RL formulation from the one explored in our work. First of all, to process the environment state, composed of the problem graph G, the hardware graph H, and the partial embedding's current status, CHARME utilizes a *Graph Neural Network* architecture. The network is trained to learn an optimal policy for *ordering* the logical variables from G. At each sequential step, the RL agent selects a single logical variable, while the subsequent task of identifying and placing the *entire corresponding chain* onto the hardware graph H is delegated to a deterministic algorithm, based on subroutines previously developed for a heuristic, see Ngo et al. (2023).

CHARME's evaluation was performed on two groups of very sparse problem graphs. The first set consisted of Barabási–Albert graphs with up to 150 nodes and a degree

of 10, resulting in a graph density of approximately 6%. The second set included 185 graphs derived from a biological optimization problem, with sizes from 6 to 165 nodes. These were partitioned by density, where the highest-density category was defined as graphs having an edge count at least three times their node count. Despite not having access to specific association between node count and density, it can be seen that the density is $\geq 31\%$ for a 20-node graph, but only $\geq 6\%$ for a 100-node graph. This evaluation context is important, as the density of the QPU hardware itself is extremely low (e.g., a Zephyr topology with 2000 qubits has a density of $\approx 1\%$). The problem graphs used to test CHARME are therefore much closer in density to the QPU topology than to the fully connected graphs common in optimization. While ME remains a challenging problem, this setting likely requires less complex and shorter qubit chains, presenting a different challenge than embedding dense graphs. Furthermore, the target hardware graphs used in their evaluation (around 16.000 qubits) were significantly larger than those on currently available quantum hardware. This suggests the method is effective at handling large hardware graphs, but leaves open the question of its ability to compress embeddings into the limited-qubit space of real-world QPUs.

3 Reinforcement learning for minor embedding

This section describes our proposed RL model, which is built on simple yet flexible architectures. The basis of our model is a RL Agent developed using a Multi-Layer Perceptron (MLP) architecture and trained with Proximal Policy Optimization (PPO) to iteratively assign a problem variable to a qubit in order to build the mapping between problem variables and hardware qubits required by minor embedding. This choice is motivated by the limited established literature on RL for minor embedding. We therefore start with a relatively simple and dependable architecture, that will allow to better understand and explore the task dynamics, its feasibility, and its challenges without confounding factors that more complex architectures would introduce. Our approach offers several practical advantages: MLPs are easy to implement and relatively fast to train, while PPO is robust, stable across a wide range of tasks, and exhibits strong empirical performance in high-dimensional action spaces which makes it well-suited for the minor embedding problem.

One disadvantage of the MLP agent architecture is that its structure does not natively allow leveraging graph properties such as permutation invariance, graph symmetries, locality, or connectivity patterns. In order to ensure that those properties are learned by the agent, we design a set of data augmentation strategies to provide the agent with symmetric variants of the same partial minor embedding encouraging a more robust and general learning process. While other architectures such as Graph Neural Networks would allow natively to better account for the graph structure, their training poses other challenges such as the need to choose aggregation functions for neighbouring nodes, the over-smoothing effects in deeper models, and a substantially more computationally expensive training in particular for dense graphs, therefore we leave them for future work.

3.1 State observation and actions

In our proposed model the agent is tasked to perform an action based on a partial minor embedding of the problem graph G (with $|G|$ nodes) onto the hardware graph H (with $|H|$ nodes) at each step t, where it receives an observation of the state s_t. The observation is a one-dimensional array, partitioned into sections, each relating to aspects of either the problem graph G or the hardware graph H. Each component corresponds to nodes of the respective graph, following a predetermined mapping. The state observation has four components: two related to the partial minor embedding and two used to restrict the action space of the agent. The two components related to the partial minor embedding are:

- **Available qubits:** This part is denoted by $S_H \in \{0,1\}^{|H|}$, with each component indicating whether that qubit in the hardware graph is available (value 1) or already assigned to a chain and therefore unavailable (value 0). At the beginning of each episode, all its values are set to 1.
- **Missing G links:** This part is denoted by $S_G \in \mathbb{Z}^{|G|}$, with each component indicating how many problem variables that node still needs to be connected with. More precisely, for every node G_j that is connected to node G_i in the problem graph, the corresponding qubit chain C_j must be connected to C_i. This measure of how many inter-chain connections are missing serves as an indication to the agent of which parts of the minor embedding are still incomplete.

When designing the action space for a RL agent it is important to ensure that the number of actions does not grow excessively with the problem size. A naive space where the agent could be tasked to select both a node from G and a corresponding node from H would result in an action space of size $|G| \times |H|$ which would rapidly become too large. In order to constrain the dimensionality of the action space to avoid combinatorial growth, we build our RL model so that the agent is provided with a problem variable and has

to identify which node of H to allocate it to. At each step t the node from G is selected with a round-robin (RR) strategy that iterates over the problem variables, then the action a_t performed by the agent only selects a node from the hardware graph H, which will be added to the chain associated to the current node from G. After the action, the next node from G is selected according to the RR sequence, only among those nodes that still lack all the required inter-chain connections.

In order to let the agent understand the RR approach, the last two components of the state observation are:

- **Current node:** Denoted by $S_R \in \{0,1\}^{|G|}$, is a one-hot encoded vector where the node from G which should be added to the minor embedding is marked with 1.
- **Chain of current node:** Denoted by $S_C \in \{0,1\}^{|H|}$, this component is a binary vector indicating which qubits belong to the chain of the current node selected from G. At the beginning of each episode, all its values are set to 0.

Furthermore, the available qubits vector S_H is additionally constrained such that only the qubits that are both available and adjacent to the chain of the currently selected G node in RR are marked as available (value 1). This results in S_H being the same mask applied via IAM (see Section 2.4.3) to the policy, as shown in Fig. 5.

An illustrative example of the entire observation vector during an intermediate minor embedding state is depicted in Fig. 6.

3.2 Reward function

After selecting an action based on the current state, the agent receives a scalar reward from the environment. The design of the reward function depends directly on the objective one

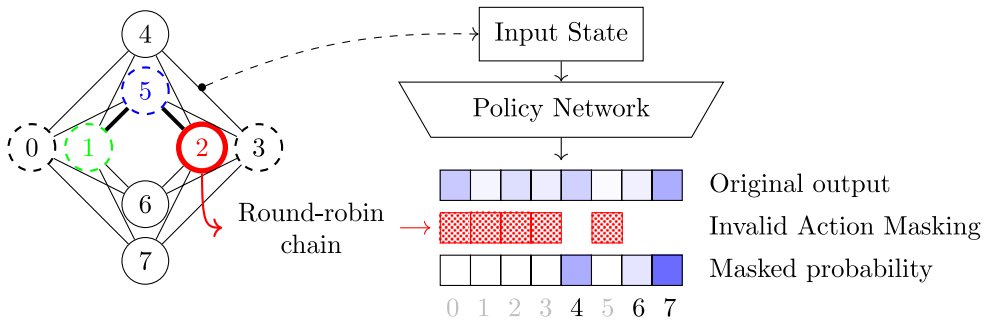


Fig. 5 Application of Invalid Action Masking to the agent's policy. On the left there is the partial minor embedding on which the agent is working, with three different chains (green, red and blue) containing respectively nodes 1, 2 and 5. This is the environment's state, whose observation is received by the agent on the right. The policy network gets the state as input and outputs a series of values, one for each possible action, i.e., nodes from 0 to 7. Since the current round-robin G node is the one corresponding to the red chain (containing only H node 2), the applied mask comprises all the nodes in the chain (2), the ones that are not adjacent to the chain (0, 1, 3) and the ones already embedded in other chains (1, 5). Therefore, the only nodes which will have a non-null value in the actual output will be 4, 6 and 7

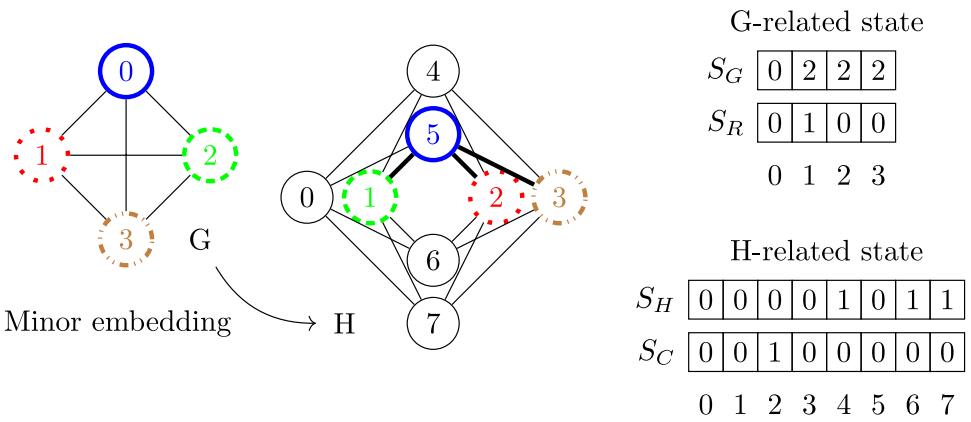


Fig. 6 Example of the state's components in an intermediate phase of the RL minor embedding procedure. The agent has to embed a G graph with 4 nodes into an H Chimera graph with 8 nodes. There are four chains already embedded in H, solid blue (5), dashed green (1), dotted red (2), dashed and dotted brown (3). The current round-robin G node is 1 (S_R). Node 0 has already completed all connections in the minor, while nodes 1, 2 and 3 are still missing 2 links (S_G). H nodes 4, 6 and 7 are available (S_H) to be added and adjacent to the chain of G node 1, composed of only H node 2 (S_C)

wants to optimize during minor embedding. Given that our main goal is to obtain a valid minor but also to minimize the length of the chains, each action gives a fixed negative reward, so that the agent is encouraged to maximize the cumulative reward by creating shorter chains. In particular, in the experiments presented in Section 5, we use a negative reward of -0.1 for each action.

At the same time, one of the advantages of the RL framework is its flexibility, which allows the reward function to be changed or redesigned depending on different optimization goals. For example, it would be possible to define a reward based on the actual solution quality of the embedded problem when executed on a quantum annealer. Another option would be to use a sparse reward scheme, where the agent receives a non-zero reward only when a complete and valid minor is produced. While these examples show the adaptability of the reward mechanisms, they would each present new challenges to address and, therefore, the exploration of these options constitutes a broad research direction that goes beyond the scope of this paper.

4 Experimental protocol

The goal of the experimental analysis is to assess the effectiveness of the proposed RL model in finding valid mappings between problem variables and hardware nodes in two scenarios: fully connected problem graphs, and randomly generated ones. In the fully connected scenario, the agent is trained to perform Minor Embedding (ME) of a specific fully connected graph G onto a specific topology graph H. In the random graph scenario, each agent is trained to perform ME of problem graphs G with different sizes and densities onto a specific topology graph H, with the objective of evaluating whether the model is capable of generalizing to unseen instances and varied topologies.

The two scenarios are tested on graphs G of different topologies and sizes. The number of nodes of the fully connected G ranges from 3 to 10, while for the randomly generated graphs we test on G with up to 30 nodes, on account of the sparser topology producing an easier minor embedding problem and allowing the agent to scale to larger problem graphs. The process used to generate the random graphs is described in Section 4.1.

For what concerns the hardware graph H, we generate them based on the number of unit cells per side, H_{size}, with two topologies: Chimera, with a number of nodes ranging from 32 to 2048 ($H_{\text{size}} \in [2, 16]$), and Zephyr, with a number of nodes ranging from 160 to 2176 ($H_{\text{size}} \in [2, 8]$). These topologies correspond to the oldest (Chimera) and newest (Zephyr) topology available at the time these experiments were conducted, and their H_{size} has been chosen to ensure the number of qubits is comparable. We should note that the size of H directly controls the number of actions available for the agent. In our setting this produces two coupled effects. First, the policy network must encode a sufficiently informative representation of a larger H. Second, training a RL agent becomes more challenging because PPO must explore a much larger action space effectively. Given that a minor embedding found on a smaller H can be seamlessly applied to any larger H with the same topology, we test the agent on multiple H_{size} to track how problem complexity scales with the action space.

4.1 Randomly generated graphs

In order to evaluate the effectiveness of our proposed RL model we train and evaluate it also on a set of randomly generated problem graphs[2]. While it could be argued that a random graph is not necessarily representative of the types of topologies and structures one could observe when applying ME to real problems, these random graphs serve the purpose of testing the RL model on a set of highly heterogeneous problems.

We generate graphs G ranging in size from 3 nodes upwards, and with varying number of edges. For a graph with $n = |G|$ nodes, the number of edges ranges from $n - 1$ to $\frac{n(n-1)}{2}$. We also ensure that all graphs are connected, i.e., from every node there should exist a path to each other node in the graph. For each number of nodes we aim to generate 1000 graphs to use during the training phase and an additional 250 to use for testing. Based on the experimental results (see Section 5) the largest problem graph tested is $n = 10$.

The process used to generate the graphs depends on their number of nodes:

- **Graphs with 3 to 5 nodes:** Starting from a fully connected graph, we generate a set of all the graphs that can be obtained from it by removing one edge. Then, for each of those graphs we repeat the process removing one further edge. We discard all graphs that have non-connected components. Due to the small number of nodes it is not possible to reach the desired number of graphs, as such the same training graph will be sampled multiple times during training.
- **Graphs with 6 to 8 nodes:** We follow the same approach used for the smaller graphs with 3 to 5 nodes, but we also ensure to keep only one instance of graphs that

[2] The dataset of randomly generated graphs is available at https://github.com/qcpolimi/RLxME_Dataset

are *isomorphic*.[3] Once this phase is completed, we obtain a set of graphs each defining an isomorphism class. Based on their number we compute how many instances for each class are required in order to reach the target number of training and testing graphs, and finally we generate them by applying a random node permutation. We always ensure that no two graphs are identical.

- **Graphs with more than 9 nodes:** For graphs of this size the exhaustive generation of both instances as well as one instance for each isomorphism class becomes unfeasible. The approach we adopt here is to first identify what is the minimum and maximum number of edges these graphs could have (i.e., $n-1$ to $\frac{n(n-1)}{2}$), then we compute how many graph instances for each of the values in this range we should generate in order to reach the target number of training and testing graphs. Given a number of edges, a new graph instance is generated at random.[4] We guarantee that the set does not contain isomorphic graphs.

The testing data is created with a 20% random holdout of the graphs, stratified on the number of nodes, hence 1000 are used for training and 250 for testing for each node size. The process accounts for the strategies adopted to construct graphs of each size: for small graphs from 3 to 5 nodes we apply a simple random holdout; for graphs of 6 to 8 nodes the random holdout is stratified on the isomorphism classes; finally, for graphs of more than 9 nodes the holdout is stratified on the number of edges in the graph. Note that while the testing data will never contain graphs that also appear in the training data, the testing data of small graphs (from 3 to 5 nodes) may contain isomorphic ones. We believe this is not a problem as it serves as a way to test whether the RL model has learned to model graph symmetries, and it only occurs for small graphs.

4.2 Training protocol

The agent is trained using Proximal Policy Optimization (see Section 2.4.2) and Invalid Action Masking (see Section 2.4.3, an example shown in Fig. 5).

In the fully connected graph scenario the agent is trained on performing minor embedding of a specific graph G on a specific topology H, therefore it only observes a single fully connected graph. The training budget is of 1 million steps, however, as we observed in the results (see Section 5.3), the agent tends to reach convergence **much faster** especially when H is small. This means that the main Reinforcement Learning (RL) interaction loop described in Section 2.4 is repeated that number of times before stopping the training phase.

In the random graph scenario the training is performed by feeding all the training graphs to the agent ordered by increasing number of nodes. The purpose of this strategy is to allow the agent to initially focus on structurally simpler instances. Given that the training process is performed on many more graphs and to ensure the agent has the time to train on each of them, the training continues until all the graphs in the dataset have been observed at least once.

During training it is guaranteed that at least 10^3 graphs for each number of nodes are used. When the number of nodes is small, i.e., 3-5, the number of existing training graphs is lower than this threshold, therefore the same training graph may be sampled multiple times. Once all graphs up to $|G| = 10$ have been sampled, the process repeats from $|G| = 3$, until the step budget is depleted.

In both scenarios the training is done on 10 independent agents each starting from a different random initialization. During testing, each agent is tasked to generate 10 minor embeddings for each test graph. In the fully connected scenario the testing graph is the same the agent has been trained on, while in the random graph scenario the testing graphs are those held-out as described in Section 4.1.

4.3 Training graph augmentations

The hardware graphs H possess several *symmetries*, stemming from their graph nature but also from the highly regular structure of the hardware topologies. These include global symmetries such as node permutations, reflections, and rotations of the layout of the hardware graph, particularly when H is formed by repeating patterns like Zephyr or Chimera unit cells. As a consequence, multiple minors can be functionally equivalent up to a relabelling or spatial transformation of the physical qubits. For example, a valid minor can be rotated $90°$ across a regular 2D layout of cells or flipped horizontally without altering its correctness or quality. This permutation invariance implies that an optimal policy for minor embedding should ideally be invariant (or at least equivariant) under these transformations.

However, MLP-based policy architectures lack any inherent mechanism to model this invariance. Since they operate on a flattened, fixed-size observation vector, the agent will consider two isomorphic states as completely different, leading to slow training and poor generalization. In order to improve the robustness of the MLP agent to those

[3] Verifying whether two graphs are isomorphic is computationally expensive. We rely on an approximate method of the networkx package which checks the degree sequence of the nodes and identifies rapidly graphs that are surely non-isomorphic based on that https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.isomorphism.faster_could_be_isomorphic.html

[4] We rely on the following function of the networkx package https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.gnm_random_graph.html

symmetries, at each step of the training process we can modify the partial minor embedding by applying an isomorphic transformation on the environment. This strategy is inspired by similar techniques in image recognition and structured decision-making problems (Shorten and Khoshgoftaar 2019; Silver et al. 2016). The core idea is to expose the agent to different but semantically equivalent versions of the environment state, so that the learned policy becomes more robust to the symmetries of the underlying problem.

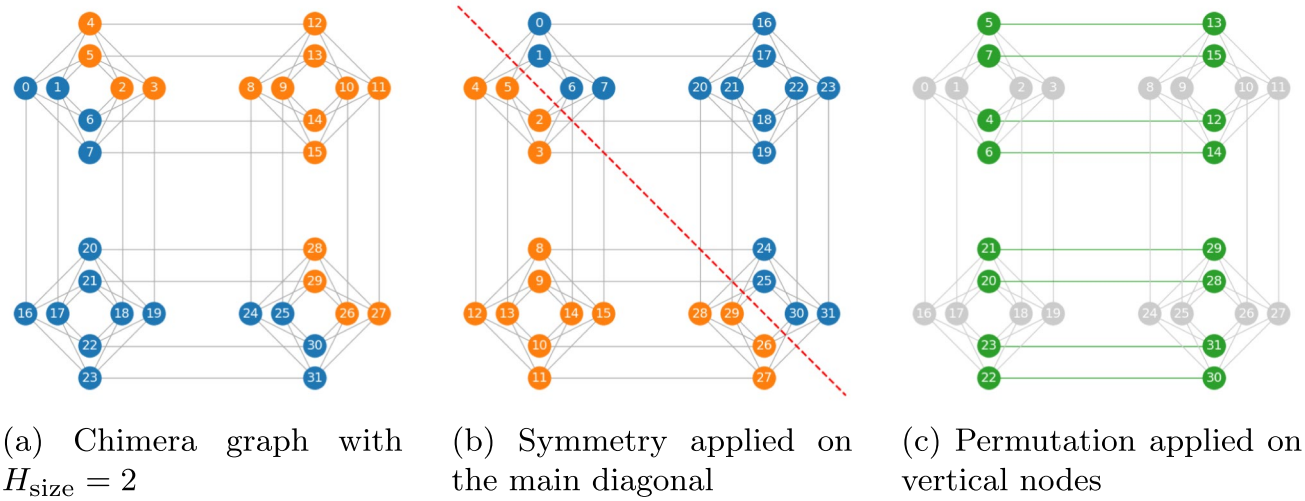
The data augmentation strategies we implemented can be grouped in three categories:

- **90° Rotations:** clockwise and counter-clockwise.
- **Mirroring:** along the vertical axis, horizontal axis, main diagonal (see Fig. 7b) and anti-diagonal.
- **Permutation:** along the vertical axis (given a row of the topology we apply the same permutation to all vertical qubits, with a different permutation applied on each row of the topology, see Fig. 7c), similarly along the horizontal axis.

This corresponds to a total of 8 possible data augmentations. At each training step, a set of zero or more augmentations, without repetitions, is selected at random and applied to the H graph. Each transformation results in a shuffled version of the state: if, for example, nodes 0 and 7 are swapped, the state vector is also updated accordingly. This process is illustrated in Fig. 8.

This strategy is conceptually aligned with successful practices in other domains, such as data augmentation in computer vision, where image transformations (e.g., flips, crops, rotations) help convolutional networks learn translational invariance. In reinforcement learning, analogous techniques have been employed to improve sample efficiency and policy generalization, especially in spatially structured environments such as robotic control or grid-based navigation tasks. In our case, the augmentation serves to regularize the policy and promote invariance to graph isomorphisms and spatial symmetries.

While effective in the small to medium-scale setting, this data augmentation approach faces inherent limitations



(a) Chimera graph with $H_{\text{size}} = 2$ (b) Symmetry applied on the main diagonal (c) Permutation applied on vertical nodes

Fig. 7 Example of augmentations on a Chimera hardware graph with $H_{\text{size}} = 2$ (4 unit cells). From the original graph (7a) we show the main diagonal symmetry (7b) and the permutation on vertical qubits (7c). Colours are used just to improve the visibility of the augmentations

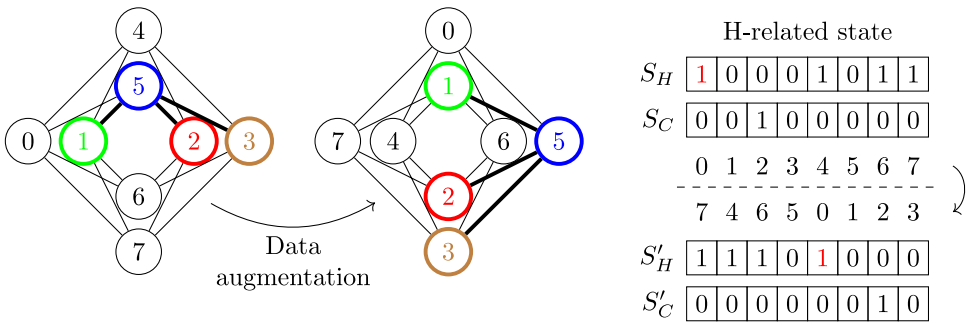


Fig. 8 Example of data augmentation on the same state shown in Fig. 6. A permutation on the horizontally-place nodes is applied after a clockwise rotation of 90°. On the right the effect of such transformation is shown on the part of the state related to H. The original values (above) are reordered accordingly to the transformation, as if they occupy the new place assigned to them (7 instead of 0, 4 instead of 1 and so on)

as the size of the hardware graph H increases. The number of possible augmentations grows rapidly, and the diversity of states the agent needs to generalize over increases combinatorially. Moreover, designing augmentation rules that remain valid and computationally tractable on large, irregular or defected hardware topologies becomes increasingly challenging. Thus, although data augmentation can improve the robustness of the MLP-based agent, especially on randomly generated graphs, it cannot fully resolve the lack of topology awareness in the architecture. While there are architectures that natively model this, such as Graph Neural Networks (GNNs), we leave their exploration as a future work.

4.4 Environment initialization

Each time the reinforcement learning environment completes an episode, its internal state, described in detail in Section 3.1, is re-initialized. This is necessary to ensure that the environment begins from a clean and valid configuration. In particular, the mask of the available qubits S_H is reset, any previous minor assignments are cleared and the round-robin pointer is set to the first node of G.

4.5 Implementation details

The reinforcement learning agent is trained using the `stable-baselines3` library (Raffin et al. 2021), which provides a robust implementation of the Proximal Policy Optimization (PPO) algorithm. The environment is built upon the `gymnasium` (Towers et al. 2024) interface and simulates the minor embedding process in order to prepare the state observation vector (see Section 3.1). The hyperparameters used for PPO are the default ones from `stable-baselines3`.

4.6 Evaluation metrics

The effectiveness of our RL model is evaluated using two main criteria: *Success Rate* and *Qubit Efficiency Ratio*, to assess its ability to generate valid minors as well as its efficiency in using the available qubits.

4.6.1 Success Rate (SR)

The *Success Rate (SR)* measures the quota of valid minors over those generated by the 10 RL agents trained on different random initializations. For the fully connected scenario, the models are tested only on 1 testing graph, for the random scenario the model is tested on 250 graphs for each number of nodes. For each testing graph, the 10 RL agents generate 10 minors each.

4.6.2 Qubit Efficiency Ratio (QER)

We also assess the quality of the minor embeddings by comparing the number of physical qubits used by the model against those used by the `minorminer` baseline. This is captured by the *Qubit Efficiency Ratio (QER)*, defined as:

$$\text{Qubit Efficiency Ratio} = \frac{|\mathcal{E}_{\text{MM}}|}{|\mathcal{E}_{\text{RL}}|}, \tag{13}$$

where $|\mathcal{E}_{\text{RL}}|$ denotes the number of qubits used by the model in its best minor embedding (i.e., the one using the fewest qubits among the different testing trials), and $|\mathcal{E}_{\text{MM}}|$ is the minimum number of qubits used by `minorminer` on the same problem graph, among 100 minors for the fully connected scenario and 10 minors for each of the testing graphs in the randomly generated graphs scenario. Given that `minorminer` typically produces highly compact minors for problems of the scale used in this work, it serves as a near-optimal baseline for this comparison. A higher QER value (i.e., closer to 1) indicates that the model's minor embedding is close in quality to the baseline. In contrast, excessive qubit usage w.r.t. to `minorminer` will increase the denominator in Eq. (13), pushing the ratio towards 0.

5 Results and discussion

In this section we present the results of the experimental analysis along the two scenarios: fully connected graphs and randomly generated ones. For the first scenario we use both topologies Chimera and Zephyr and discuss in details the results on each of them. For the random graph scenario we instead focus on Zephyr.

5.1 Fully connected problem graphs

5.1.1 Chimera topology

Chimera is the oldest topology where each qubit is connected only to up to 6 other ones, as such it represents a hardware graph where the minor embedding will tend to be rather large. A selection of the results for the Chimera topology is reported in Table 1, the full results are reported in Appendix A (Tables 5 and 6).

Success rate First, we will discuss the results of the base version of the agent, without data augmentations. By analysing the success rate of the minor embedding, i.e., the quota of minor embeddings generated by the agent that constitute a correct and complete minor of graph G on the hardware graph H, we can see how for small H_{size} (2 to 6) the success

Table 1 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on fully connected problem graphs G mapped to Chimera topology graphs H. This table shows a slice of the full results, which are reported in Appendix A (Tables 5 and 6)

$\mathbf{H_{size}}$	DA	$\lvert\mathbf{G}\rvert = 4$		$\lvert\mathbf{G}\rvert = 6$		$\lvert\mathbf{G}\rvert = 8$		$\lvert\mathbf{G}\rvert = 10$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
2	✓	100	7 ± 3	100	14 ± 0	100	23 ± 2	0	—
		100	6 ± 1	100	14 ± 0	98	23 ± 2	0	—
4	✓	100	6 ± 1	100	14 ± 1	100	29 ± 6	1	115 ± 0
		100	7 ± 2	100	17 ± 9	100	42 ± 20	0	—
6	✓	100	9 ± 2	100	22 ± 5	10	73 ± 2	0	—
		100	10 ± 11	100	64 ± 62	24	257 ± 18	0	—
8	✓	100	15 ± 5	100	61 ± 25	1	490 ± 0	0	—
		100	32 ± 37	97	206 ± 149	5	439 ± 51	0	—
10	✓	100	23 ± 8	97	308 ± 176	0	—	0	—
		100	119 ± 126	77	593 ± 148	0	—	0	—
12	✓	100	96 ± 148	75	617 ± 231	0	—	—	—
		100	294 ± 161	69	961 ± 138	0	—	—	—
14	✓	100	70 ± 116	66	1081 ± 158	0	—	—	—
		100	412 ± 212	69	1323 ± 166	1	1182 ± 0	—	—
16	✓	100	167 ± 251	70	1328 ± 314	2	1674 ± 129	—	—
		100	642 ± 292	50	1451 ± 292	0	—	—	—

rate is very high for $|G| \leq 8$. As H_{size} increases we can see that the success rate drops sharply for larger G, to the point of failing to produce a valid minor at $H_{\text{size}} = |G| = 8$. Instead, for smaller G the success rate remains rather stable, for example, $|G| = 6$ has a success rate above 90% for $H_{\text{size}} \leq 10$ and then a success rate oscillating between 60% and 75% as H grows larger up to 16. These results indicate that the success rate is much more sensitive to the size of G than it is to the size of H. This effect is immediately visible from Fig. 9. The reasons for this can be several, but it is apparent that, as G becomes larger, the agent struggles to model the increased complexity of the minor embedding process with many increasingly long chains that need to be connected across several cells. Which points to the need to strengthen the ability of the agent to model complex graph structures.

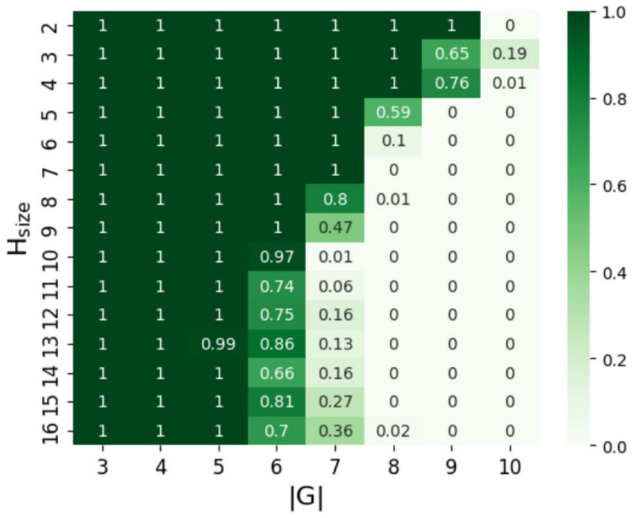


Fig. 9 Success rate for the fully connected scenario on Chimera

The data augmentation aims to tackle this limitation. Since the agent is provided with graphs and minors that have been permuted, in different ways, the aim is that this should improve the ability of the agent to identify the right correlations in the data and learn a more robust latent representation of the overall minor structure. If we look at the results in Table 1 we can compare the success rate of the base model with that where the data augmentation is applied. The results are mixed. If we go back to the previous example of $|G| = 6$, the success rate of the base model is above 90% for $H_{\text{size}} \leq 10$ but that of the data augmentation version begins to drop earlier, with $H_{\text{size}} = 10$ having a success rate of 77%. For larger H the success rate oscillates, as in the base model, but not consistently. For example, for $H_{\text{size}} = 14$ the base version has a success rate of 66% while the data augmentation version of 69%. On the opposite end is the immediately following $H_{\text{size}} = 16$ where the base version has a success rate of 70% while the data augmentation version of 50%, a 20 points difference. While this seems to suggest the data augmentation is not beneficial, in this experiment at least, as G becomes larger the conclusions change. If we look at $|G| = 7$ (see Table 6) the base version of the agent starts to struggle with $H_{\text{size}} = 8$ and becomes largely ineffective for larger H oscillating between a success rate of 1% and 36%. However, it is here that we see the benefit of the data augmentation which is able to keep the success rate

higher for intermediate values of H_{size}. For example, for $H_{\text{size}} = 10$ the base version has a success rate of 1% while the data augmentation version of 32%, or for $H_{\text{size}} = 12$ where the base version has a success rate of 16% while the data augmentation version of 30%. However, these gains do not remain present at even higher H. For the largest G, the data augmentation is sometimes the only version that is able to provide any minor embedding at all, in particular for $H_{\text{size}} = 7, 9, 11, 14$ where the base version always fails. A possible explanation for this effect is that, since the agent already finds embedding fully connected graphs challenging, the additional variability introduced by data augmentation further strains it. We will see instead, when presenting the results on the randomly generated graphs, that data augmentation is more consistently helpful as H_{size} increases. Overall, these results indicate that on fully connected graphs data augmentation is not a consistently better strategy, and point to the possibility of treating it as a hyperparameter to be activated or not based on the results of the model.

Number of qubits Another important dimension to consider is how many qubits are used by a successful minor embedding. Clearly, the first aim is to obtain a valid minor, however once that has been achieved, smaller minor embeddings are preferable. This is due to a number of factors but mainly to how larger minor embeddings require longer chains of connected qubits. In practice, the physical consequence of a long chain is that it is going to be more difficult for the qubits to change their value as doing so requires to overcome an energy barrier that does not only depend on the other chains the qubit is connected to, but also on all the other qubits of the same chain that are strongly coupled to each other. Table 1 shows the average and standard deviation of the number of qubits required by the successful minor embeddings produced by the RL agent. If we focus on small H_{size} we can see how the number of qubits does not change significantly with increasing sizes of G. For example, when $H_{\text{size}} = 4$ a graph $|G| = 4$ with the base agent requires 6 qubits, while a graph $|G| = 6$ requires 14 and finally $|G| = 8$ requires 29. This indicates that the agent is rather efficient in this setting. If we look at increasing sizes of H, however, we can see a marked increase in the number of required qubits, which surpasses 1000. The first question to ask is why should it be necessary to use a higher number of qubits to embed the same graph on a larger hardware graph. The answer is that it should not. This effect shows that while the agent is able to successfully generate valid minor embeddings with a very high number of qubits, which is in the same order of the qubits existing on the available quantum annealers, over a thousand, it is struggling to correctly model and explore this larger hardware graph in a way that produces an efficient minor. The limited modelling capacity of the MLP-based agent is also compounded by the known difficulties of RL on large action spaces, which would begin to play a role for large H graphs. Fortunately, as previously mentioned, a minor embedding found on a smaller H_{size} can be used as is on a larger one. Given that, as H_{size} increases, training the agent becomes more challenging, a simple strategy to avoid unnecessary complexity is to generate the embedding on a smaller H_{size} and then increase it only if the problem graph G is too large to be successfully embedded. We should note however that different agent architectures may allow to mitigate this effect.

If we compare the effect of the data augmentation, again we can see that no consistent pattern emerges, with a few exceptions. For $|G| = 4$ the number of qubits required by the augmented versions is consistently higher than the base version, sometimes by a considerable margin. Consider for example $H_{\text{size}} = 16$, if we look at $|G| = 4$ we can see how the base version requires only an average of 167 qubits, while the data augmentation version requires 642. If we move to larger G this difference disappears. When $H_{\text{size}} = 16$ and $|G| = 6$ the base version requires approximately 1300 qubits while the data augmentation one approximately 1450. Then, the difference becomes smaller and smaller as G becomes larger. The reason for this effect may be explained by how for large H and relatively small G the base version of the agent finds easier to use one cell or its immediate surrounding ones, while the augmented ones are pushed to explore the hardware graphs much more, hitting into the limited modelling capacity of this architecture and resulting in a successful, but inefficient, minor embedding.

Figure 10 shows a visualization of the Qubit Efficiency Ratio, the ratio between the number of qubits required by

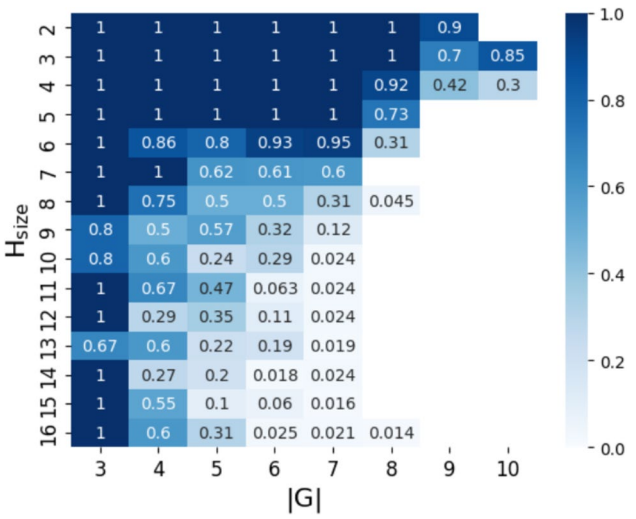


Fig. 10 Qubit Efficiency Ratio for the fully connected scenario on Chimera

the minor embeddings produced by `minorminer` versus the ones identified by the agent without augmentations. We can see how for $|G| \leq 7$ and for small H graphs (up to 200 qubits) the agent uses as many qubits as `minorminer`. This is typically due to the fact that the minor embedding is either optimal or very close to optimal. As the size of G increases we can see that the minor embedding becomes less efficient and the agent finds minors requiring from slightly more to up to three times more qubits. A much more marked effect is present by increasing the size of H while keeping G smaller. As we discussed previously, this is a region where the agent is less efficient, with the augmented versions requiring a very large number of qubits compared to the base version. The base version itself however already requires many more qubits than `minorminer`. For example, for $H_{\text{size}} = 12$ and $|G| = 6$ the agent requires ten times as many qubits as `minorminer`, and almost fifty times as many when $|G| = 7$. This again points to how the modelling capability of the agent and the large action space make this a challenging scenario.

5.1.2 Zephyr topology

Zephyr is the newest topology where each qubit is connected to up to 20 other ones, as such it represents a hardware graph where the minor embedding will tend to be rather compact w.r.t. Chimera. A selection of the results for the Zephyr topology are reported in Table 2, the full results are reported in Appendix A (Tables 7 and 8).

Success rate The first observation we can draw is that all experiments exhibit a success rate of 100% both increasing the size of G and by increasing the size of H. It should also be noted that the size of cells in Zephyr is larger than in Chimera, so $H_{\text{size}} = 8$ for Zephyr is comparable to $H_{\text{size}} = 16$ for Chimera.

The 100% success rate is also maintained for the data augmentation version, indicating that while it does not, and indeed could not, further improve the success rate, it is not detrimental. This stands in contrast to what observed for Chimera where its impact was sometimes positive and other times negative.

The results clearly show that the RL agent is very effective for this type of setting.

Number of qubits By comparing the number of qubits required for a successful minor embedding we can see how it remains low for the base version of the agent when minor embedding graphs on small $H_{\text{size}} \leq 4$. For $H_{\text{size}} = 4$ a graph $|G| = 4$ requires only 7 qubits, while a graph $|G| = 6$ requires 13 and finally $|G| = 8$ requires 22. While the number of qubits required increases faster than the size of the graph G, this increase is limited. If we consider small $|G| \leq 4$ we can see how the number of qubits required remains limited as the size of H increases, i.e., for $|G| = 4$ the minor embedding requires 7 qubits on $H_{\text{size}} = 4$, 13 on $H_{\text{size}} = 6$, and 21 on $H_{\text{size}} = 8$. So again increasing the size of H results in a minor embedding that is less efficient,

Table 2 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on fully connected problem graphs G mapped to Zephyr topology graphs H. This table shows a slice of the full results, which are reported in Appendix A (Tables 7 and 8)

$\mathbf{H_{size}}$	DA	$\mathbf{\lvert G\rvert = 4}$		$\mathbf{\lvert G\rvert = 6}$		$\mathbf{\lvert G\rvert = 8}$		$\mathbf{\lvert G\rvert = 10}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
2	✓	100	4 ± 0	100	9 ± 1	100	14 ± 6	100	18 ± 1
		100	4 ± 0	100	9 ± 1	100	18 ± 11	100	23 ± 11
3		100	5 ± 3	100	9 ± 2	100	15 ± 3	100	20 ± 3
	✓	100	5 ± 5	100	12 ± 9	100	40 ± 28	100	105 ± 56
4		100	7 ± 4	100	13 ± 2	100	22 ± 7	100	32 ± 20
	✓	100	6 ± 2	100	29 ± 27	100	158 ± 98	100	319 ± 71
5		100	10 ± 5	100	26 ± 8	100	45 ± 17	100	165 ± 72
	✓	100	9 ± 19	100	88 ± 72	100	394 ± 111	100	532 ± 141
6		100	13 ± 7	100	49 ± 49	100	213 ± 102	100	724 ± 135
	✓	100	21 ± 32	100	307 ± 189	100	540 ± 181	100	861 ± 148
7		100	15 ± 11	100	72 ± 63	100	473 ± 117	100	1014 ± 111
	✓	100	96 ± 125	100	434 ± 167	100	921 ± 190	100	1093 ± 195
8		100	21 ± 24	100	172 ± 159	100	742 ± 144	100	1260 ± 285
	✓	100	169 ± 163	100	594 ± 195	100	1040 ± 244	99	1459 ± 316

which can be explained by the difficulty of the agent to correctly model the larger hardware graph and possibly by the detrimental effect caused by the increased action space the agent needs to explore.

By increasing both G and H we see that the minor embedding starts to require a very large number of qubits beyond $H_{\text{size}} = |G| = 8$, with a minor of almost 750 qubits, again suggesting that the MLP-based agent struggles when the topology of the graph becomes complex and, despite being able to provide a successful minor embedding, this comes with an inefficient use of the available qubits.

By looking at the effect of the data augmentation we can see a consistent increase in the number of qubits required. This difference depends on the scenario. For small $|G| \leq 4$ the data augmentation requires a similar number of qubits until $H_{\text{size}} \geq 7$, where it starts to increase steeply. See for example $|G| = 4$ where its minor embedding requires 13 qubits with the base variant and 21 with the data augmentation one when $H_{\text{size}} = 6$, while it requires 15 with the base version and a much larger 96 with the data augmentation one for the larger $H_{\text{size}} = 7$. The difference becomes even more marked for $H_{\text{size}} = 8$. This again confirms that the architecture of the MLP-based agent struggles with modelling minor embeddings on larger graphs and the data augmentation strains the model ability further. A similar observation is valid also by restricting to lower $H_{\text{size}} \leq 5$ and increasing the size of G. With $H_{\text{size}} = 5$ the minor embedding of $|G| = 4$ requires 10 qubits with the base agent and 9 with the data augmentation one. This rare instance where the data augmentation requires fewer qubits can be explained with its very high variance in many experiments. If however we move to larger G the discrepancy again increases in favour of the base model. For $|G| = 6$ the base model requires 26 qubits while the data augmentation version requires 88 and finally for $|G| = 8$ the base version requires 45 qubits while the data augmentation around 400, almost ten times as many. A curious pattern is that as both H and G grow past 7, the discrepancy reduces substantially. For the larger experiment where $H_{\text{size}} = |G| = 8$ the base version requires almost 750 qubits while the data augmentation version around 1000. A possible explanation for this effect is that in this setting the base agent too needs to explore a much larger portion of the graph H and so it has to learn a more difficult problem, whereas for smaller ones the base model can solve a simpler problem compared to the agent that uses data augmentation.

Figure 11 shows a visualization of the Qubit Efficiency Ratio. For almost all the sizes of G embedded on $H_{\text{size}} \leq 4$ (around 500 qubits) the agent is able to use a number of qubits that is either identical or rather close to

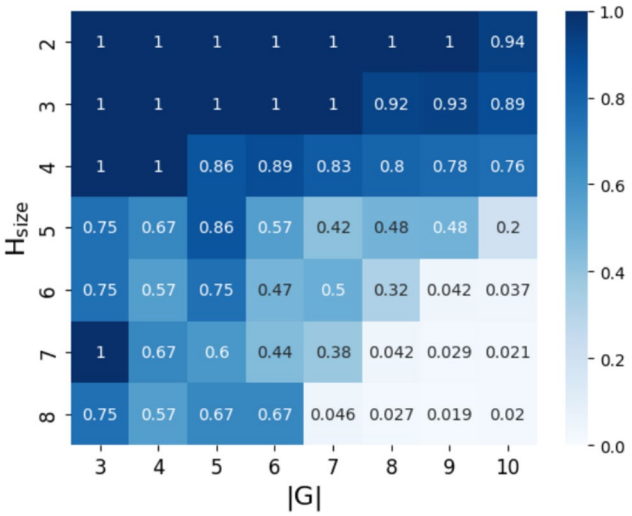


Fig. 11 Qubit Efficiency Ratio for the fully connected scenario on Zephyr

what is required by `minorminer`. This is a similar finding as in Chimera and again is due to how both `minorminer` and the agent are able to find minor embeddings that are either optimal or very close to optimal.

A similar observation can be made if we compare the number of qubits for small $|G| \leq 4$ embedded on increasingly larger hardware graphs. There the minor embedding is somewhat less efficient with `minorminer` being able to use two-thirds of the qubits required by the RL agent. The smaller number of qubits required by the minor embedding can be attributed to the higher connectivity of Zephyr which results in shorter chains and therefore requires a less complex modelling of the embedded graph topology.

However, when embedding graphs with larger $|G|$ and H_{size} the minor embedding becomes very inefficient. For example, for $|G| = 9$ and $H_{\text{size}} = 6$ the minor produced by the agent requires around 25 times as many qubits as `minorminer`. Again, the impact of this effect is relatively small as the results show that those graphs can be easily and efficiently embedded in smaller H graphs and so, one can use smaller ones as target and move to larger ones only if the minor embedding fails. This mitigation strategy would alleviate this problem until graph G is large enough to require a larger H as well.

5.1.3 Topology impact on minor embedding

Figure 12 shows a direct comparison of how the agent behaves when embedding graphs of increasing size onto Chimera and Zephyr topologies. The most important difference between the two topologies is the connectivity, which

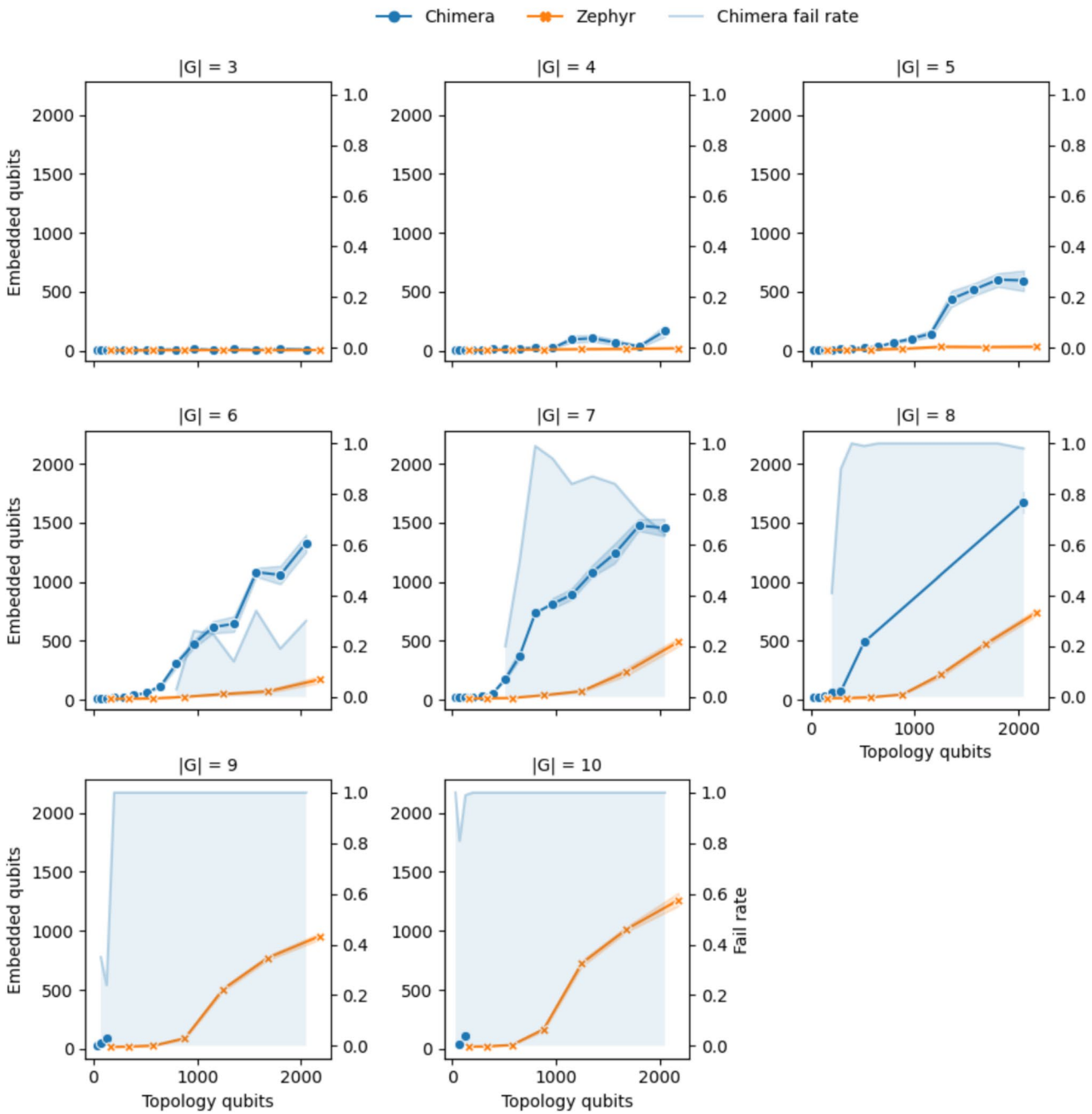


Fig. 12 Number of qubits used by the RL agent to embed fully connected graphs of varying sizes $|G|$ onto Chimera and Zephyr topologies with increasing numbers of hardware qubits (to allow a comparison between the two hardware topologies). Each subplot corresponds to a fixed $|G|$, showing the qubit usage for both topologies, along with the minor embedding failure rate on Chimera (on the right y-axis, shaded area). While both topologies perform similarly for small $|G|$, Chimera exhibits higher qubit usage and increasing failure rates for larger graphs. Zephyr consistently achieves successful minor embeddings with fewer qubits and no observed failures across all tested scenarios

for Chimera is 6 while for Zephyr is of 20. As such, we expect that, given the same G, the minor embedding's size on Zephyr will be much smaller than on Chimera and the chains much shorter. This will result in an easier minor embedding especially as the size of G increases, which is indeed what we can see. For smaller graphs, particularly when $|G|$ is 3, 4, or 5, both Chimera and Zephyr perform similarly, with low qubit usage and no noticeable failures. However, as the the size of G increases beyond this point, some trends start to emerge.

Table 3 Comparison of the number of qubits required for the minor embedding of random graphs built by the RL agent (trained on graphs up to $|G| = 10$), when using data augmentations during training only, and using them both during training and testing. The **DA** column indicates whether data augmentation was used only during training (Train) or during both training and testing (Test)

$\mathbf{H_{size}}$	DA	$\mathbf{\lvert G\rvert = 3}$	$\mathbf{\lvert G\rvert = 4}$	$\mathbf{\lvert G\rvert = 5}$	$\mathbf{\lvert G\rvert = 6}$	$\mathbf{\lvert G\rvert = 7}$	$\mathbf{\lvert G\rvert = 8}$	$\mathbf{\lvert G\rvert = 9}$	$\mathbf{\lvert G\rvert = 10}$
2	Train	3 ± 0	5 ± 1	6 ± 1	8 ± 1	10 ± 1	12 ± 2	14 ± 4	19 ± 9
	Test	3 ± 0	5 ± 1	6 ± 1	8 ± 1	10 ± 1	12 ± 1	14 ± 2	16 ± 3
3	Train	3 ± 0	5 ± 1	6 ± 1	8 ± 2	11 ± 3	19 ± 13	37 ± 25	64 ± 36
	Test	3 ± 1	5 ± 1	6 ± 1	8 ± 1	10 ± 1	12 ± 2	14 ± 2	17 ± 5
4	Train	3 ± 1	5 ± 1	7 ± 1	11 ± 6	29 ± 19	54 ± 31	89 ± 44	129 ± 59
	Test	3 ± 1	5 ± 1	7 ± 1	8 ± 1	10 ± 1	12 ± 2	14 ± 3	18 ± 6
5	Train	4 ± 1	7 ± 7	11 ± 13	27 ± 29	54 ± 45	95 ± 63	150 ± 87	208 ± 115
	Test	4 ± 1	5 ± 1	7 ± 1	9 ± 2	11 ± 3	13 ± 5	16 ± 7	20 ± 11
6	Train	5 ± 3	9 ± 10	16 ± 20	47 ± 44	80 ± 69	114 ± 92	156 ± 119	202 ± 146
	Test	4 ± 1	6 ± 1	8 ± 4	10 ± 6	13 ± 8	16 ± 11	20 ± 17	25 ± 22
7	Train	4 ± 1	8 ± 8	32 ± 39	85 ± 68	149 ± 94	216 ± 131	304 ± 173	400 ± 226
	Test	4 ± 1	6 ± 1	7 ± 2	9 ± 5	12 ± 7	15 ± 12	19 ± 19	26 ± 32
8	Train	4 ± 1	20 ± 39	53 ± 85	131 ± 132	224 ± 179	317 ± 220	437 ± 266	556 ± 316
	Test	4 ± 1	6 ± 2	8 ± 5	10 ± 9	14 ± 16	18 ± 24	24 ± 42	36 ± 68

In the case of Chimera, the number of qubits required grows more quickly, especially from $|G| = 6$ onwards. Along with the rise in qubit usage, the failure rate also starts to increase and becomes prominent from $|G| = 7$ to $|G| = 10$. For the largest graphs, the failure rate in Chimera reaches high levels, and in many cases, minor embeddings are no longer consistently successful. This confirms how the minor embedding process becomes more difficult or less reliable as the complexity of the problem graph increases.

Zephyr, on the other hand, shows a different pattern. The qubit usage grows more slowly with graph size and remains lower than in Chimera for the same $|G|$ and number of qubits. More notably, the RL agent does not encounter failures when embedding into Zephyr, even as the problem size grows. This more stable behaviour indicates that the model can reliably find minor embeddings across all tested graph sizes when working with the Zephyr topology.

Overall, while both topologies appear sufficient for smaller graphs, Zephyr tends to support minor embeddings more efficiently and consistently as graph complexity increases. In contrast, ME on Chimera is more challenging and the agent shows signs of strain both in terms of resource requirements and reliability when faced with larger or more connected graphs. This is likely due to the higher connectivity offered by the Zephyr topology w.r.t. Chimera, which makes the minor embedding problem easier by reducing the need to form long and complex chains in the minor. This is expected to improve the efficiency of minor embedding in general and, indeed, we observe that the agent's architectural limitations are mitigated by the increased hardware connectivity, exhibiting high Success Rate and requiring a number of qubits closer to that of `minorminer`.

5.2 Randomly generated problem graphs

The previous scenario focused on the minor embedding of a fully connected G, which constitutes a particularly challenging problem where the embedded graph will be very large and complex. This second scenario aims to test the effectiveness of the RL agent on the randomly generated problem graphs described in Section 4.1. Given that Zephyr is the most recent architecture, we only use this topology for this experiment. Since our MLP agent requires to fix a priori the maximum size of the graph G that can be minor embedded, and that the agent is able to scale to larger problem graphs when these are randomly generated rather than fully connected, we run multiple experiments by training the agent with graphs up to $|G| = 10, 20$, and 30.[5] For space reasons here we present a summary, the full results are available in Appendix B.

First, we discuss the impact of using the data augmentation mechanism presented in Section 4.3 at different stages. Table 3 shows the result of two different sets of experiments for the agent trained on graphs up to $|G| = 10$, one is using data augmentations only during training, testing the agent on a fixed H graph for all action steps, while the other uses data augmentations also during the testing phase. As we can see, testing on a fixed H graph results in rather inefficient minor embedding and seems to confuse the agent, consistently to what we observed in the fully connected scenario, while testing on augmented H graphs, with a different

[5] Note that the agent trained on graphs of up to $|G| = 30$ is only evaluated up to $H_{\text{size}} = 5$ on account for the large number of combinations it produces and that larger H have been already analysed in multiple previous experiments.

permutation of the original graph at each action step, is very effective and helps the agent to find a substantially smaller minor embedding. For example, when $H_{\text{size}} = |G| = 4$ both versions produce minor embeddings requiring 5 qubits, when $H_{\text{size}} = |G| = 6$ using data augmentation during testing produces minors of 10 qubits while not using it increases the minor's size to 47, and when $H_{\text{size}} = |G| = 8$ this difference becomes even more pronounced with a minor embedding requiring 18 qubits if using augmentations during testing and 317 if not. Based on this observation, in the remainder of this section we will discuss results that either do not apply data augmentation in any form, or they do *both in training and testing*.

The summary of the results for the agent trained on graphs up to $|G| = 30$ is shown in Table 4, the full results are available in Appendix B (see Table 13 and Table 14), including those for agents trained on graphs up to $|G| = 20$ and 10. Overall, the agent shows a smooth increase in the number of qubits required as $|G|$ grows. For example, on $H_{\text{size}} = 2$ the minor of $|G| = 10$ requires approximately 20 qubits, which become about 60 for $|G| = 20$ and 100 for $|G| = 30$. The increase with respect to H_{size} behaves differently, it is limited when going from $H_{\text{size}} = 2$ to 3, while it approximately doubles when going from $H_{\text{size}} = 3$ to 4 and triples from $H_{\text{size}} = 4$ to 5. Hence, we observe again that significantly increasing the number of available actions for the agent, as well as the complexity of the target graph, results in a more inefficient minor. It should also be noted that the size of H is generally much larger than the problem graph G and, since the number of qubits grows quadratically with the number of cells, small increases in H_{size} have a more pronounced impact than those on G. If we consider the impact of data augmentation, we observe that it tends to reduce significantly the number of qubits required when H_{size} is large, typically by a factor of two to three, while it does not have a consistent effect on smaller instances. This is an indication that when the model is able to generate good minors, the additional variability introduced by the data augmentation is helpful to improve the learning process and the efficiency.

Comparing the Success Rate, the agent achieves a very high rate in most cases, only beginning to drop beyond $|G| = 25$ for $H_{\text{size}} = 4$ and $|G| = 20$ for $H_{\text{size}} = 5$, and still exceeding 80% for $|G| = 30$ and $H_{\text{size}} \in 3, 4$ when using data augmentation. It is interesting to observe how the success rate drops more strongly on $H_{\text{size}} = 2$, which can be explained by the minor running out of available qubits in the target architecture and approaching the limits of what can fit on it. Approaching this limit strains the model because it requires to tightly pack the minor and thus a highly efficient qubit allocation. The data augmentation seems to have two different effects, for the small $H_{\text{size}} = 2$ and G large enough to approach the limits of what can fit on H, the introduction of data augmentation does not affect the number of qubits required by the minor but reduces significantly the success rate. This effect is not present for $H_{\text{size}} = 3$ and the opposite occurs at $H_{\text{size}} \in 4, 5$, where data augmentation yields a substantially higher success rate than the version without it. This suggests that data augmentation works well when the minor embedding does not need to be tightly packed and the agent has not reached the limits of its ability to handle the task. As previously mentioned, a simple and practical strategy to mitigate larger minors as H_{size} increases is to construct them on smaller H_{size} and increase the size only

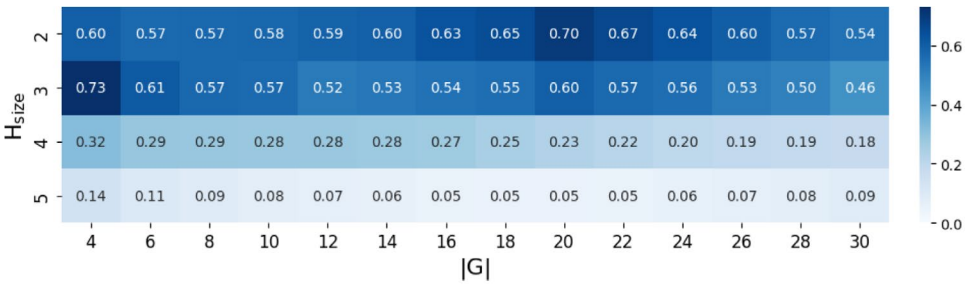
Table 4 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent on the randomly generated problem graphs G mapped to Zephyr topology graphs H (with$H_{\text{size}} \in [2, 5]$on the columns). The agent is trained on graphs of up to$|G| = 30$

$\|\mathbf{G}\|$	**DA**	$\mathbf{H_{size} = 2}$		$\mathbf{H_{size} = 3}$		$\mathbf{H_{size} = 4}$		$\mathbf{H_{size} = 5}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
5	✓	100	9 ± 2	100	8 ± 1	100	17 ± 11	100	44 ± 33
		100	8 ± 2	100	8 ± 2	100	9 ± 2	100	20 ± 24
10	✓	100	21 ± 4	100	22 ± 4	100	44 ± 20	100	164 ± 85
		100	22 ± 4	100	21 ± 3	100	23 ± 5	100	57 ± 64
15	✓	100	37 ± 6	100	42 ± 9	100	82 ± 45	99	403 ± 168
		100	43 ± 15	100	37 ± 5	100	39 ± 6	99	129 ± 167
20	✓	100	55 ± 9	100	65 ± 16	97	158 ± 107	41	593 ± 171
		88	67 ± 23	100	56 ± 6	100	59 ± 12	83	148 ± 175
25	✓	95	81 ± 17	99	95 ± 27	78	189 ± 102	13	607 ± 142
		65	87 ± 23	100	79 ± 15	100	92 ± 37	72	184 ± 177
30	✓	44	108 ± 20	85	142 ± 47	50	262 ± 108	3	668 ± 108
		23	105 ± 24	85	131 ± 50	83	179 ± 103	48	253 ± 189

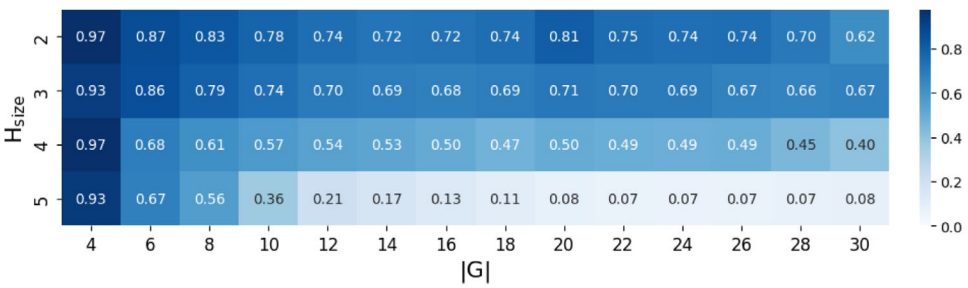
if necessary. Within this approach, using data augmentation can help maintain a smaller number of qubits even when H is larger than necessary, which simplifies the choice.

The results discussed so far refer to the agent trained to embed graphs up to $|G| = 30$, and we have also trained the agent on smaller cases $|G| = 10$ (see Tables 9 and 10) and $|G| = 20$ (see Tables 11 and 12). The previous observations apply consistently to both cases. The number of qubits required grows smoothly as both G and H_{size} increase, but without the acceleration at $H_{\text{size}} = 4$ previously observed, which may be due to the easier minor embedding task. Similarly, the Success Rate remains very high, mostly at 100%, with only a small reduction for the model trained on graphs up to $|G| = 20$ for the largest graphs and $H_{\text{size}} = 8$ when no data augmentation is used, while still remaining above 80%. The Success Rate of the version using data augmentation is always 100%. As a final comparison, consider the number of qubits required to embed the largest graphs used to train the smaller model: for $|G| = 10$, the model trained on graphs up to $|G| = 10, 20$ and 30 yields very similar results when using data augmentation, whereas the version without augmentation requires more qubits when trained on larger graphs. This behavior is consistent when comparing the minors for $|G| = 20$ obtained with the models trained up to $|G| = 20$ and 30.

Now that we have discussed the results of the agent along different directions, we compare its QER with `minorminer` and contrast the results with what we observed in the fully connected graph scenario. The result of this comparison, when the agent is trained without data augmentation, are reported in Fig. 13. From a high level, the two scenarios (fully connected and random graphs) exhibit a similar overall pattern, with an optimal or near-optimal minor embedding for small G and small H. The differences, as well as the effect of data augmentations, become more pronounced as both sizes increase. If we compare the QER of the highest values for $|G|$ and H_{size} we can see that they are on the same order of magnitude in both scenarios (0.02 for $|G| = 10, H_{\text{size}} = 8$ on the fully connected graphs and 0.09 for $|G| = 30, H_{\text{size}} = 5$ on the random graphs). This is similarly reflected when the agent is trained on smaller random graphs up to $|G| = 10$ and 20 (see Figs. 17 and 19 in the Appendix). Again, this points to the challenging task of modelling effectively the topology of both graphs as well as of the incomplete minor embedding that is being built by the RL agent, to decide which should be the most effective action. Furthermore, the experiments on the random problem graphs do not show the sharp increase in the number of qubits that was present in fully connected graphs, where in some scenarios the number of qubits jumped by a factor of 10 by simply increasing either $|G|$ or H_{size} by 1. For example, with $H_{\text{size}} = |G| = 7$, the QER for the minor embedding of the fully connected G was 0.38 while the slightly larger $|G| = 8$ had 0.04. This points to a crucial threshold where the modelling becomes very ineffective. This threshold does not appear to be present for the minor embedding of random problem graphs, likely due to how they are less connected and the resulting minor is simpler, pushing this crucial threshold towards larger H and G. This also suggests



(a) Average QER



(b) Best QER

Fig. 13 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 30$, with no data augmentation. Full results are shown in Fig. 21

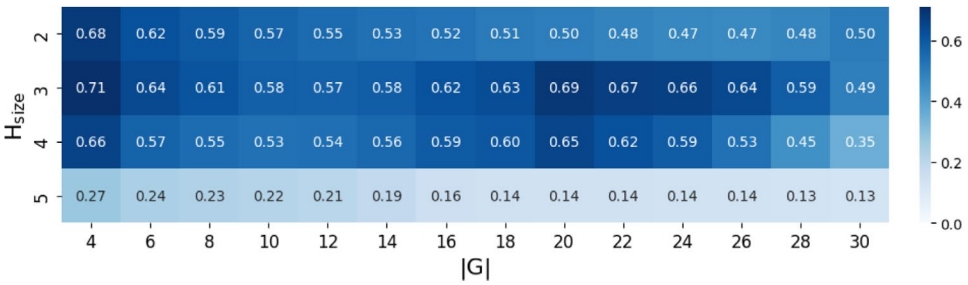
that training the agent on a variety of graphs with different densities can be helpful to stabilize the training when the task's complexity increases. However, the large difference between the Average and Best QER, especially for lower $|G|$ and H_{size}, indicates that different runs and graphs have a very high variance and therefore, even though the QER is good, the training is more unstable in those regions.

Another direction we can analyse is to assess if training the agent for graphs up to different sizes affects its QER, and we find that it does. If we look at lower sizes such as $H_{\text{size}} = 5$, we can see that the QER of the agent trained on fully connected graphs is higher than that of the agent trained on random graphs up to $|G| = 30$. However, the opposite happens when training the agent on random graphs up to $|G| = 10$ and 20, where the QER is always higher compared to the fully connected scenario. Observing that the QER in certain regions is systematically shifted to higher or lower levels, suggests that increasing the graph size is affecting the agent in multiple ways. In particular, the more the agent trains on larger graphs, the lower its QER will be on smaller graphs. This observation may be particularly important to support the scaling of the agent to even larger G as it may require to refine the training protocol so that the agent does not forget how to generate good minors of small graphs. We leave exploring this direction as future work.

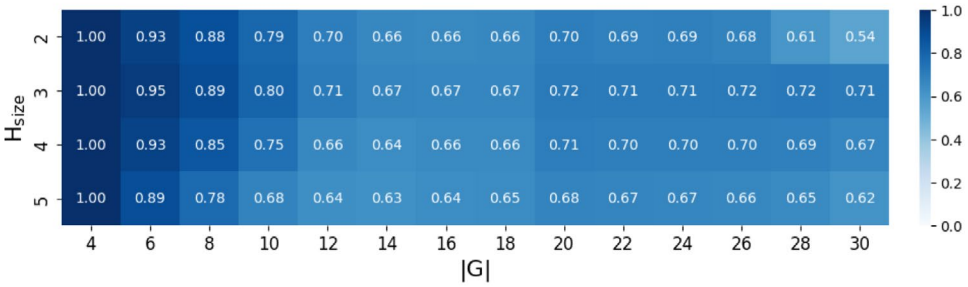
The Average and Best QER of the RL agent when trained and tested using data augmentation are shown in Fig. 14. While using data augmentations lowers the Average QER on small H and G ($H_{\text{size}} = 2, 3$, $|G| = 4, 6$), the overall results show a clear improvement both in training and testing as the graphs become larger. The Average QER doubles across almost every experiment on larger H graphs ($H_{\text{size}} = 4, 5$) and the Best QER dramatically increases, demonstrating that data augmentation helps the agent better understand the involved structures of at least a portion of the random graph dataset. For example, on $H_{\text{size}} = 3$ the Best QER is consistently above 0.70 from $|G| = 20$ to $|G| = 30$, indicating how the agent is very close to `minorminer`. Furthermore, the Best QER only slightly decreases as H_{size} increases, confirming how the data augmentations are a useful strategy to reduce the number of qubits required when the hardware graphs become larger. Nonetheless, as H becomes larger the distance between Best and Average QER increases, indicating how the model is again starting to struggle and generating an increasing number of minors that are valid, but require a larger number of qubits. Similar tests, using the data augmentation mechanism also during the testing phase, were conducted on the fully connected problems, but did not show any relevant difference w.r.t. when used only during training. This may be due to the regular structure of the problem graphs being learned in that scenario. Overall, these results indicate that while the data augmentation strategies did not appear effective on fully connected graphs, it is very useful when applied during both training and testing on randomly generated graphs.

5.3 Training stability and convergence

Training RL agents is a notoriously challenging task due to the inherent complexity of the underlying optimization



(a) Average QER



(b) Best QER

Fig. 14 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 30$ using data augmentation both in training and testing. Full results are shown in Fig. 22

landscape. Unlike supervised learning, where the objective function is typically convex or at least stationary, RL loss surfaces are often *non-convex*, *non-smooth*, and *non-stationary*. The agent's actions influence its environment, which in turn modifies future observations and rewards, yielding a moving target for the optimizer. Furthermore, in actor-critic setups such as the one employed in this work, the interaction between the policy and the value function can lead to phenomena like *policy collapse*, *vanishing gradients*, or *catastrophic forgetting*, especially when rewards are sparse or delayed. As a consequence, convergence is not guaranteed, and instabilities are often observed in training dynamics. Monitoring learning curves over long training horizons becomes essential to assess whether the agent is progressively improving or being misled by spurious feedback.

In this work, the reward function was designed to be both dense and aligned with the minor embedding objective: the agent receives a fixed negative reward at each step. This formulation encourages the agent to minimize the number of actions taken to construct a valid minor, thereby promoting the formation of shorter chains. The use of immediate, consistent penalties avoids the difficulties associated with sparse or delayed rewards. While alternative rewards such as energy-based or sparse terminal rewards could be explored in future work, this simple step-based penalty shows to be sufficiently stable and effective.

Figures 15 and 16 show the convergence behaviour of the RL agent trained on the same fully connected problem graph but targeting two different Zephyr graphs, respectively with $H_{\text{size}} = 2$ and $H_{\text{size}} = 8$ unit cells. Each curve represents the mean and standard deviation of 10 runs of problem graphs G of different sizes, reporting the mean episode length over episode batches. Episode length corresponds to the number of actions taken by the agent before the episode terminates, either by successfully constructing a complete minor embedding or by failing to produce a valid one.

In this context, shorter episode lengths typically indicate more efficient minor embedding policies because the agent

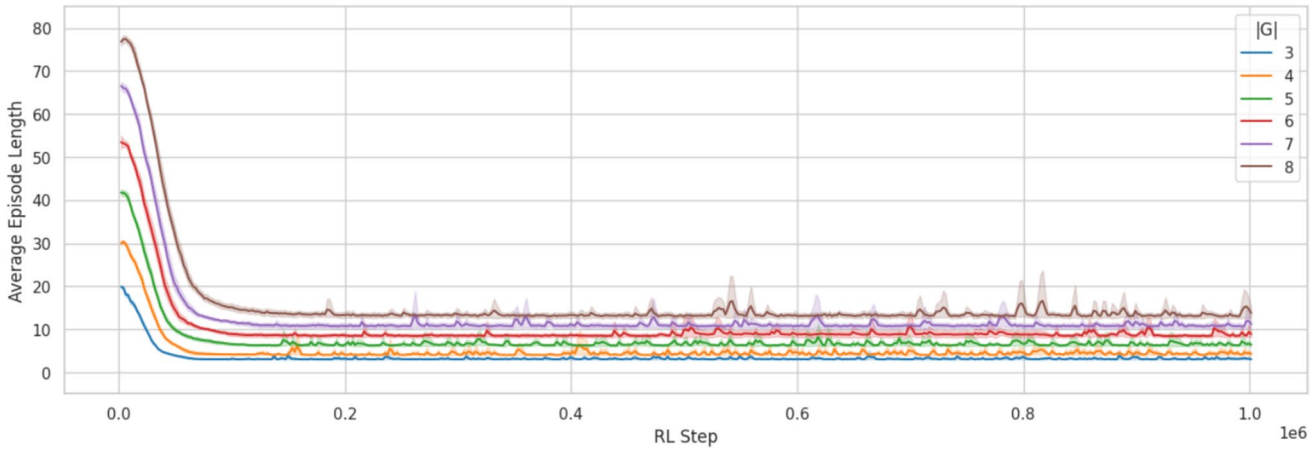


Fig. 15 Convergence of the RL agent training on Zephyr with $H_{\text{size}} = 2$ in the fully connected graphs scenario

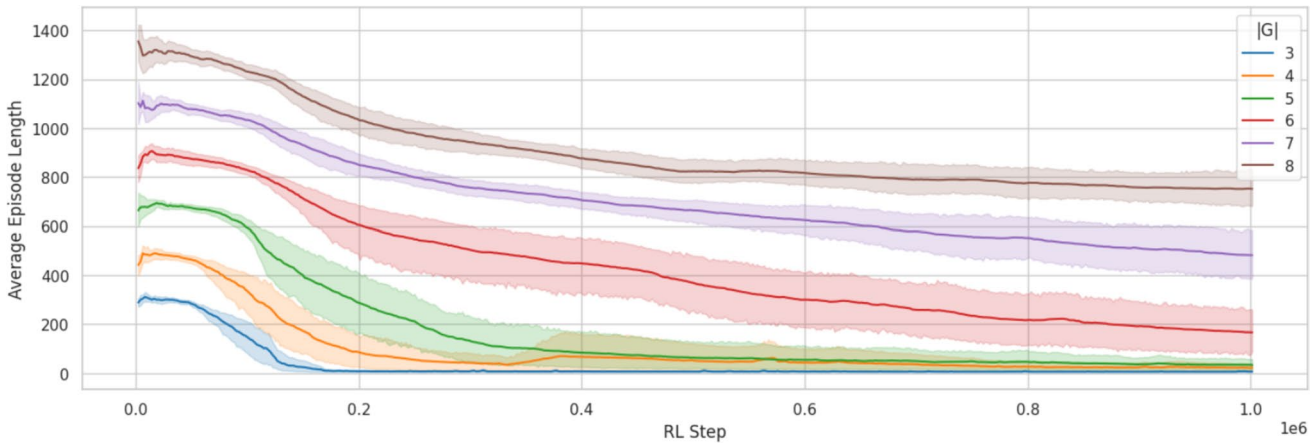


Fig. 16 Convergence of the RL agent training on Zephyr with $H_{\text{size}} = 8$ in the fully connected graphs scenario

finds a solution in fewer steps. Hence, improvements in the agent can be inferred from a consistently declining episode length that stabilizes at a value that should be the optimal number of qubits required for the minor embedding.

In the $H_{\text{size}} = 2$ setting (Fig. 15), corresponding to a relatively small hardware topology (160 qubits), the agent displays robust convergence across all G sizes and random initializations. Episode lengths drop rapidly during the early phase of training (first 100k steps), stabilizing to relatively low values. This suggests that the agent learns a consistent policy for constructing valid minors in a compact search space.

In contrast, the $H_{\text{size}} = 8$ setting (Fig. 16) involves a significantly larger and more complex hardware graph (2176 qubits), resulting in longer episodes and a more challenging minor embedding task. Nonetheless, the majority of G show clear evidence of learning, episode lengths initially increase or plateau, possibly due to exploration noise or suboptimal early policies, before steadily decreasing over the course of training. The final episode lengths vary across runs, indicating some residual variance, but in all successful runs the trend is downward. One or two trajectories for higher G sizes stagnate at higher values, reflecting partial convergence or suboptimal local minima, again consistent with known RL difficulties. Overall, however, the agent exhibits promising training dynamics even in a high-dimensional action space, indicating that the policy architecture and training setup support meaningful learning.

Taken together, these results suggest that the agent is capable of learning effective minor embedding policies for moderately sized Zephyr topologies. While occasional seed failures or plateaus are to be expected in this setting, the general trend across both experiments demonstrates that the agent is able to cope with increasing hardware complexity without suffering from severe instability or divergence.

6 Conclusion

In this work we proposed a Reinforcement Learning agent trained with Proximal Policy Optimization to tackle the problem of identifying the mapping between problem variables and hardware qubits required for Minor Embedding, with a goal of exploring machine learning methods that would offer a higher degree of flexibility compared to traditional heuristics.

The MLP-based reinforcement learning agent demonstrated the ability to generate valid minor embeddings of several hundreds and sometimes more than a thousand of qubits, a scale comparable with today's existing quantum annealers. The quota of valid minor embeddings for fully connected graphs tends to decrease for the larger graphs on the older Chimera topology while it remains high, and much more stable, on the newer Zephyr topology, indicating the agent was able to benefit from the higher qubit connectivity. On randomly generated graphs the agent was similarly able to generate valid minor embeddings spanning several hundreds of qubits and exhibited again stable behaviour as both the size of the problem graph and the hardware increase. On the other hand, the agent was less efficient than `minorminer` in reducing the number of required qubits in particular on the more challenging fully connected graphs, indicating that further work is required to improve the efficiency of RL based methods.

The use of data augmentations in the form of random graph permutations proved beneficial for randomly generated graphs when used both during the training and testing phases, resulting in a considerable reduction of the number of qubits required by the minor embedding, but was less beneficial when the agent was reaching the limits of its modelling ability or when the problem graph was approaching the limits of what could be embedded on a small hardware.

Overall, while our results indicate that the use of RL agents for minor embedding is possible even with relatively simple architectures, they also point to limitations of the agent which struggles to model the topology of the graphs, limiting its ability to fully exploit structural regularities in the problem and hardware graphs. Based on this, an important future direction is to explore the use of other architectures such as Graph Neural Networks (GNNs) which natively model the characteristics of hardware structures in a way that could make the training more efficient and the agent more robust but would also require to address limitations of GNNs related to over-smoothing, choosing appropriate aggregation functions and account for their much more expensive training time.

Appendix A Full results for the fully connected graphs

This section reports the full results for the minor embedding of fully connected graphs on the Chimera topology (see Table 5 for $|G| = 3$ to 6, and Table 6 for $|G| = 7$ to 10) and the Zephyr topology (see Table 7 for $H_{\text{size}} = 2$ to 5, and Table 8 for $H_{\text{size}} = 6$ to 8).

Table 5 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on fully connected problem graphs G (from$|G| = 3$to 6) mapped to Chimera topology graphs H

$\mathbf{H_{size}}$	**DA**	$\|\mathbf{G}\| = \mathbf{3}$		$\|\mathbf{G}\| = \mathbf{4}$		$\|\mathbf{G}\| = \mathbf{5}$		$\|\mathbf{G}\| = \mathbf{6}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
2	✓	100	4 ± 0	100	7 ± 3	100	8 ± 0	100	14 ± 0
		100	4 ± 0	100	6 ± 1	100	8 ± 0	100	14 ± 0
3	✓	100	4 ± 1	100	6 ± 0	100	8 ± 0	100	14 ± 2
		100	4 ± 0	100	6 ± 2	100	8 ± 0	100	15 ± 1
4	✓	100	5 ± 2	100	6 ± 1	100	10 ± 5	100	14 ± 1
		100	4 ± 1	100	7 ± 2	100	9 ± 4	100	17 ± 9
5	✓	100	6 ± 4	100	7 ± 1	100	11 ± 3	100	18 ± 3
		100	5 ± 5	100	6 ± 0	100	9 ± 8	100	20 ± 18
6	✓	100	5 ± 1	100	9 ± 2	100	19 ± 11	100	22 ± 5
		100	5 ± 1	100	10 ± 11	100	28 ± 33	100	64 ± 62
7	✓	100	5 ± 2	100	12 ± 3	100	21 ± 6	100	38 ± 21
		100	5 ± 2	100	10 ± 12	100	51 ± 71	100	77 ± 88
8	✓	100	8 ± 3	100	15 ± 5	100	24 ± 8	100	61 ± 25
		100	5 ± 8	100	32 ± 37	100	151 ± 105	97	206 ± 149
9	✓	100	9 ± 5	100	19 ± 8	100	36 ± 23	100	118 ± 68
		100	6 ± 10	100	15 ± 35	100	130 ± 138	75	464 ± 122
10	✓	100	9 ± 4	100	23 ± 8	100	70 ± 39	97	308 ± 176
		100	20 ± 36	100	119 ± 126	100	241 ± 191	77	593 ± 148
11	✓	100	13 ± 7	100	24 ± 12	100	101 ± 88	74	476 ± 149
		100	64 ± 83	100	263 ± 130	97	502 ± 180	61	778 ± 93
12	✓	100	7 ± 3	100	96 ± 148	100	144 ± 160	75	617 ± 231
		100	99 ± 98	100	294 ± 161	99	622 ± 240	69	961 ± 138
13	✓	100	12 ± 5	100	106 ± 152	99	438 ± 335	86	645 ± 317
		100	220 ± 113	100	369 ± 234	97	853 ± 203	67	1048 ± 191
14	✓	100	8 ± 3	100	70 ± 116	100	518 ± 277	66	1081 ± 158
		100	238 ± 180	100	412 ± 212	100	977 ± 232	69	1323 ± 166
15	✓	100	13 ± 10	100	36 ± 57	100	603 ± 290	81	1057 ± 362
		100	238 ± 191	100	459 ± 248	100	1046 ± 253	58	1450 ± 229
16	✓	100	10 ± 4	100	167 ± 251	100	596 ± 417	70	1328 ± 314
		100	319 ± 207	100	642 ± 292	96	1000 ± 281	50	1451 ± 292

Table 6 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on fully connected problem graphs G (from$|G| = 7$to 10) mapped to Chimera topology graphs H

$\mathbf{H_{size}}$	DA	$\|\mathbf{G}\| = \mathbf{7}$		$\|\mathbf{G}\| = \mathbf{8}$		$\|\mathbf{G}\| = \mathbf{9}$		$\|\mathbf{G}\| = \mathbf{10}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
2	✓	100	18 ± 1	100	23 ± 2	100	29 ± 0	0	—
		98	18 ± 1	98	23 ± 2	76	29 ± 1	0	—
3	✓	100	20 ± 6	100	26 ± 8	65	49 ± 11	19	43 ± 4
		100	20 ± 2	100	25 ± 3	76	53 ± 9	2	69 ± 3
4	✓	100	20 ± 2	100	29 ± 6	76	88 ± 19	1	115 ± 0
		100	26 ± 17	100	42 ± 20	64	88 ± 19	0	—
5	✓	100	27 ± 12	59	63 ± 33	0	—	0	—
		100	38 ± 33	78	104 ± 51	5	167 ± 22	0	—
6	✓	100	32 ± 21	10	73 ± 2	0	—	0	—
		99	93 ± 67	24	257 ± 18	0	—	0	—
7	✓	100	54 ± 18	0	—	0	—	0	—
		83	176 ± 108	14	347 ± 19	0	—	0	—
8	✓	80	176 ± 92	1	490 ± 0	0	—	0	—
		56	333 ± 165	5	439 ± 51	0	—	0	—
9	✓	47	370 ± 141	0	—	0	—	0	—
		48	558 ± 50	11	559 ± 62	1	599 ± 0	0	—
10	✓	1	736 ± 0	0	—	0	—	0	—
		32	691 ± 54	0	—	0	—	0	—
11	✓	6	815 ± 59	0	—	—	—	—	—
		28	830 ± 59	6	834 ± 63	—	—	—	—
12	✓	16	892 ± 100	0	—	—	—	—	—
		30	928 ± 116	0	—	—	—	—	—
13	✓	13	1082 ± 86	0	—	—	—	—	—
		25	1087 ± 100	1	1216 ± 0	—	—	—	—
14	✓	16	1240 ± 180	0	—	—	—	—	—
		9	1303 ± 130	1	1182 ± 0	—	—	—	—
15	✓	27	1477 ± 134	0	—	—	—	—	—
		12	1466 ± 124	0	—	—	—	—	—
16	✓	36	1455 ± 234	2	1674 ± 129	—	—	—	—
		10	1673 ± 190	0	—	—	—	—	—

Table 7 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the fully connected graphs G (from $|G| = 3$ to 10 on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [2, 5]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\|G\|$	DA	$H_{\text{size}} = 2$		$H_{\text{size}} = 3$		$H_{\text{size}} = 4$		$H_{\text{size}} = 5$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	3 ± 0	100	3 ± 0	100	4 ± 2	100	5 ± 2
		100	3 ± 0	100	3 ± 0	100	3 ± 0	100	4 ± 2
4	✓	100	4 ± 0	100	5 ± 3	100	7 ± 4	100	10 ± 5
		100	4 ± 0	100	5 ± 5	100	6 ± 2	100	9 ± 19
5	✓	100	6 ± 1	100	8 ± 2	100	9 ± 2	100	16 ± 5
		100	7 ± 1	100	7 ± 0	100	10 ± 6	100	52 ± 70
6	✓	100	9 ± 1	100	9 ± 2	100	13 ± 2	100	26 ± 8
		100	9 ± 1	100	12 ± 9	100	29 ± 27	100	88 ± 72
7	✓	100	11 ± 3	100	13 ± 2	100	16 ± 4	100	39 ± 18
		100	12 ± 4	100	23 ± 18	100	85 ± 83	100	225 ± 141
8	✓	100	14 ± 6	100	15 ± 3	100	22 ± 7	100	45 ± 17
		100	18 ± 11	100	40 ± 28	100	158 ± 98	100	394 ± 111
9	✓	100	15 ± 2	100	18 ± 5	100	26 ± 22	100	88 ± 41
		100	20 ± 11	100	91 ± 53	100	274 ± 75	100	487 ± 104
10	✓	100	18 ± 1	100	20 ± 3	100	32 ± 20	100	165 ± 72
		100	23 ± 11	100	105 ± 56	100	319 ± 71	100	532 ± 141

Table 8 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the fully connected graphs G (from $|G| = 3$ to 10 on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [6, 8]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\|G\|$	DA	$H_{\text{size}} = 6$		$H_{\text{size}} = 7$		$H_{\text{size}} = 8$	
		SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	6 ± 1	100	5 ± 2	100	6 ± 3
		100	7 ± 13	100	16 ± 36	100	61 ± 100
4	✓	100	13 ± 7	100	15 ± 11	100	21 ± 24
		100	21 ± 32	100	96 ± 125	100	169 ± 163
5	✓	100	33 ± 50	100	31 ± 24	100	34 ± 39
		100	117 ± 76	100	208 ± 176	100	406 ± 237
6	✓	100	49 ± 49	100	72 ± 63	100	172 ± 159
		100	307 ± 189	100	434 ± 167	100	594 ± 195
7	✓	100	74 ± 44	100	239 ± 171	100	490 ± 188
		100	387 ± 201	100	655 ± 222	100	788 ± 190
8	✓	100	213 ± 102	100	473 ± 117	100	742 ± 144
		100	540 ± 181	100	921 ± 190	100	1040 ± 244
9	✓	100	504 ± 92	100	771 ± 105	100	954 ± 140
		100	645 ± 178	100	1109 ± 203	100	1287 ± 239
10	✓	100	724 ± 135	100	1014 ± 111	100	1260 ± 285
		100	861 ± 148	100	1093 ± 195	99	1459 ± 316

Appendix B Full results for the randomly generated graphs

This section reports the full results for the minor embedding of randomly generated graphs on the Zephyr topology when the agent is trained on graphs of up to $|G| = 10$ (see Tables 9 and 10), up to $|G| = 20$ (see Tables 11 and 12), and up to $|G| = 30$ (see Tables 13 and 14). We also report the Qubit Efficiency Ratio (QER) when training the agent on random graphs without data augmentation and when using data augmentation in both training and testing, up to $|G| = 10$ (Figs. 17 and 18), up to $|G| = 20$ (Figs. 19 and 20), and up to $|G| = 30$ (Figs. 21 and 22).

Table 9 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 10 nodes (shown on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [2, 5]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\lvert G \rvert$	DA	$\mathbf{H_{size} = 2}$		$\mathbf{H_{size} = 3}$		$\mathbf{H_{size} = 4}$		$\mathbf{H_{size} = 5}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	3 ± 0	100	3 ± 1	100	4 ± 1	100	5 ± 2
		100	3 ± 0	100	3 ± 1	100	3 ± 1	100	4 ± 1
4	✓	100	4 ± 0	100	5 ± 1	100	5 ± 1	100	7 ± 2
		100	5 ± 1	100	5 ± 1	100	5 ± 1	100	5 ± 1
5	✓	100	7 ± 3	100	6 ± 1	100	7 ± 1	100	8 ± 2
		100	6 ± 1	100	6 ± 1	100	7 ± 1	100	7 ± 1
6	✓	100	9 ± 3	100	8 ± 1	100	9 ± 2	100	10 ± 2
		100	8 ± 1	100	8 ± 1	100	8 ± 1	100	9 ± 2
7	✓	100	11 ± 3	100	10 ± 1	100	11 ± 3	100	13 ± 4
		100	10 ± 1	100	10 ± 1	100	10 ± 1	100	11 ± 3
8	✓	100	12 ± 4	100	12 ± 2	100	18 ± 23	100	15 ± 9
		100	12 ± 1	100	12 ± 2	100	12 ± 2	100	13 ± 5
9	✓	100	14 ± 4	100	16 ± 10	100	33 ± 43	100	25 ± 31
		100	14 ± 2	100	14 ± 2	100	14 ± 3	100	16 ± 7
10	✓	100	17 ± 7	100	26 ± 20	100	62 ± 69	100	51 ± 66
		100	16 ± 3	100	17 ± 5	100	18 ± 6	100	20 ± 11

Table 10 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 10 nodes (shown on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [6, 8]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\lvert G \rvert$	DA	$\mathbf{H_{size} = 6}$		$\mathbf{H_{size} = 7}$		$\mathbf{H_{size} = 8}$	
		SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	6 ± 2	100	6 ± 3	100	7 ± 3
		100	4 ± 1	100	4 ± 1	100	4 ± 1
4	✓	100	9 ± 5	100	8 ± 2	100	8 ± 3
		100	6 ± 1	100	6 ± 1	100	6 ± 2
5	✓	100	11 ± 6	100	10 ± 3	100	11 ± 3
		100	8 ± 4	100	7 ± 2	100	8 ± 5
6	✓	100	13 ± 6	100	14 ± 15	100	13 ± 4
		100	10 ± 6	100	9 ± 5	100	10 ± 9
7	✓	100	16 ± 8	100	19 ± 25	100	17 ± 13
		100	13 ± 8	100	12 ± 7	100	14 ± 16
8	✓	100	21 ± 12	100	24 ± 37	100	23 ± 24
		100	16 ± 11	100	15 ± 12	100	18 ± 24
9	✓	100	35 ± 37	100	50 ± 67	100	37 ± 49
		100	20 ± 17	100	19 ± 19	100	24 ± 42
10	✓	100	89 ± 99	100	107 ± 116	100	107 ± 125
		100	25 ± 22	100	26 ± 32	100	36 ± 68

Table 11 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 20 nodes (shown on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [2, 5]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\lvert G \rvert$	DA	$\mathbf{H_{size} = 2}$		$\mathbf{H_{size} = 3}$		$\mathbf{H_{size} = 4}$		$\mathbf{H_{size} = 5}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	4 ± 1	100	4 ± 1	100	5 ± 2	100	6 ± 1
		100	4 ± 1	100	4 ± 1	100	4 ± 1	100	4 ± 1
4	✓	100	6 ± 2	100	5 ± 1	100	7 ± 3	100	8 ± 3
		100	6 ± 2	100	5 ± 1	100	5 ± 1	100	6 ± 1
5	✓	100	8 ± 2	100	7 ± 1	100	10 ± 3	100	12 ± 4
		100	7 ± 2	100	7 ± 1	100	7 ± 1	100	7 ± 1
6	✓	100	11 ± 3	100	10 ± 2	100	13 ± 4	100	15 ± 4
		100	9 ± 2	100	9 ± 1	100	9 ± 1	100	10 ± 2
7	✓	100	13 ± 3	100	12 ± 2	100	16 ± 3	100	18 ± 4
		100	11 ± 2	100	11 ± 2	100	11 ± 1	100	12 ± 2
8	✓	100	15 ± 4	100	14 ± 2	100	19 ± 4	100	21 ± 5
		100	13 ± 2	100	13 ± 2	100	13 ± 2	100	14 ± 2
9	✓	100	18 ± 4	100	17 ± 2	100	22 ± 4	100	25 ± 5
		100	16 ± 2	100	16 ± 2	100	16 ± 2	100	17 ± 3
10	✓	100	20 ± 5	100	20 ± 3	100	25 ± 4	100	29 ± 6
		100	18 ± 3	100	18 ± 3	100	18 ± 3	100	19 ± 3
11	✓	100	23 ± 5	100	23 ± 4	100	28 ± 4	100	32 ± 7
		100	21 ± 3	100	21 ± 3	100	21 ± 3	100	22 ± 3
12	✓	100	25 ± 5	100	25 ± 4	100	31 ± 4	100	37 ± 9
		100	24 ± 3	100	24 ± 3	100	24 ± 4	100	25 ± 4
13	✓	100	27 ± 5	100	28 ± 4	100	34 ± 5	100	41 ± 10
		100	27 ± 4	100	27 ± 4	100	28 ± 5	100	29 ± 6
14	✓	100	30 ± 5	100	31 ± 4	100	37 ± 5	100	47 ± 11
		100	31 ± 4	100	31 ± 4	100	31 ± 6	100	32 ± 7
15	✓	100	33 ± 5	100	34 ± 5	100	41 ± 6	100	53 ± 14
		100	34 ± 5	100	34 ± 5	100	35 ± 7	100	36 ± 12
16	✓	100	36 ± 5	100	38 ± 5	100	46 ± 9	100	60 ± 19
		100	38 ± 5	100	38 ± 6	100	39 ± 11	100	41 ± 18
17	✓	100	38 ± 5	100	41 ± 6	100	51 ± 11	100	67 ± 25
		100	42 ± 6	100	42 ± 7	100	43 ± 15	100	46 ± 23
18	✓	100	41 ± 5	100	44 ± 7	100	56 ± 15	100	77 ± 37
		100	46 ± 7	100	46 ± 10	100	48 ± 23	100	52 ± 34
19	✓	100	45 ± 7	100	50 ± 12	100	63 ± 24	100	90 ± 54
		100	51 ± 8	100	52 ± 14	100	55 ± 34	100	60 ± 46
20	✓	100	52 ± 12	100	61 ± 29	100	85 ± 54	100	126 ± 102
		100	58 ± 11	100	62 ± 23	100	69 ± 54	100	76 ± 70

Table 12 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 20 nodes (shown on the rows) mapped to Zephyr topology graphs H (with $H_{\text{size}} \in [6, 8]$ on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\lvert G \rvert$	DA	$H_{\text{size}} = 6$		$H_{\text{size}} = 7$		$H_{\text{size}} = 8$	
		SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	8 ± 3	100	9 ± 3	100	16 ± 19
		100	4 ± 1	100	4 ± 1	100	4 ± 1
4	✓	100	12 ± 4	100	12 ± 4	100	41 ± 76
		100	6 ± 1	100	6 ± 1	100	6 ± 1
5	✓	100	21 ± 19	100	16 ± 4	100	82 ± 159
		100	8 ± 2	100	8 ± 2	100	8 ± 2
6	✓	100	31 ± 37	100	19 ± 5	100	97 ± 182
		100	10 ± 2	100	10 ± 2	100	10 ± 2
7	✓	100	47 ± 70	100	22 ± 5	100	129 ± 231
		100	12 ± 2	100	12 ± 2	100	12 ± 2
8	✓	100	55 ± 88	100	25 ± 6	100	155 ± 278
		100	15 ± 3	100	15 ± 2	100	14 ± 3
9	✓	100	69 ± 115	100	29 ± 6	100	183 ± 323
		100	18 ± 3	100	17 ± 3	100	17 ± 3
10	✓	100	80 ± 135	100	31 ± 7	100	226 ± 404
		100	20 ± 4	100	20 ± 3	100	20 ± 5
11	✓	100	91 ± 157	100	34 ± 8	100	259 ± 460
		100	23 ± 5	100	23 ± 6	100	23 ± 7
12	✓	100	102 ± 178	100	38 ± 9	100	283 ± 494
		100	26 ± 7	100	27 ± 9	100	27 ± 10
13	✓	100	114 ± 198	100	45 ± 16	100	314 ± 543
		100	30 ± 9	100	30 ± 12	100	31 ± 15
14	✓	100	126 ± 213	100	52 ± 22	100	344 ± 581
		100	34 ± 13	100	35 ± 18	100	35 ± 22
15	✓	100	137 ± 225	100	59 ± 29	99	407 ± 605
		100	39 ± 21	100	40 ± 26	100	40 ± 31
16	✓	99	147 ± 238	100	67 ± 36	98	429 ± 629
		100	44 ± 28	100	46 ± 40	100	48 ± 52
17	✓	97	143 ± 218	100	72 ± 41	97	432 ± 631
		100	49 ± 34	100	53 ± 52	100	55 ± 62
18	✓	95	141 ± 193	100	82 ± 56	94	444 ± 657
		100	58 ± 52	100	63 ± 68	100	67 ± 89
19	✓	94	148 ± 176	100	104 ± 84	91	448 ± 659
		100	68 ± 73	100	74 ± 87	100	84 ± 128
20	✓	92	178 ± 172	100	145 ± 148	84	398 ± 613
		100	88 ± 108	100	95 ± 120	100	118 ± 197

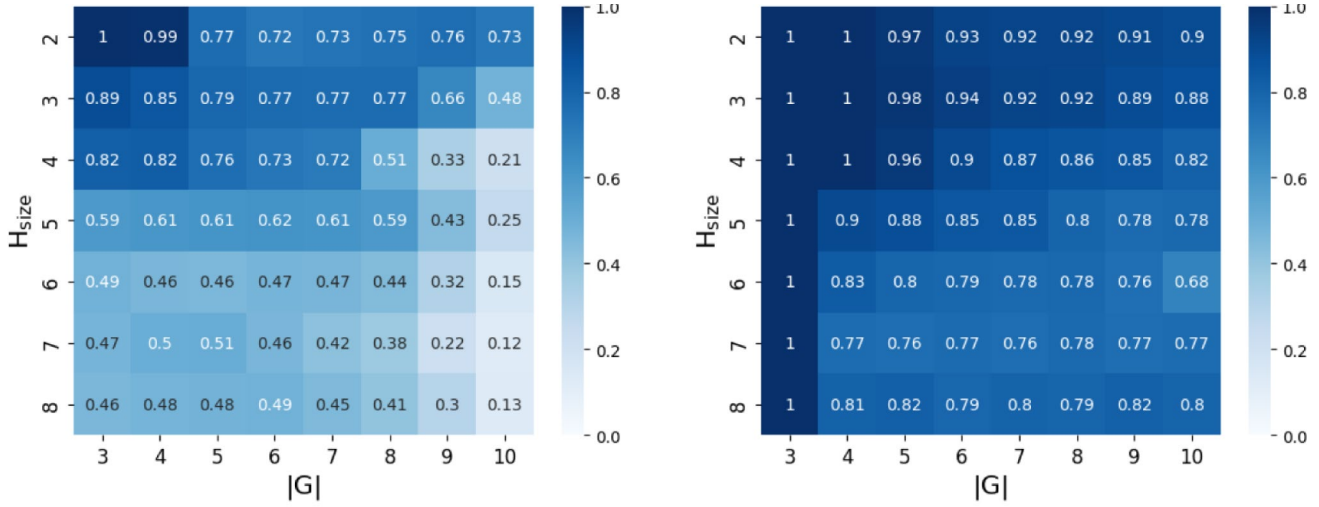
Table 13 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 30 nodes (shown from$|G| = 3$to 20 on the rows) mapped to Zephyr topology graphs H (with$H_{\text{size}} \in [2, 5]$on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

$\|\mathbf{G}\|$	**DA**	$\mathbf{H_{size} = 2}$		$\mathbf{H_{size} = 3}$		$\mathbf{H_{size} = 4}$		$\mathbf{H_{size} = 5}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
3	✓	100	5 ± 1	100	4 ± 1	100	9 ± 8	100	15 ± 11
		100	4 ± 1	100	4 ± 1	100	4 ± 1	100	10 ± 12
4	✓	100	7 ± 2	100	5 ± 1	100	13 ± 9	100	29 ± 24
		100	6 ± 1	100	6 ± 1	100	6 ± 2	100	15 ± 18
5	✓	100	9 ± 2	100	8 ± 1	100	17 ± 11	100	44 ± 33
		100	8 ± 2	100	8 ± 2	100	9 ± 2	100	20 ± 24
6	✓	100	11 ± 2	100	10 ± 2	100	22 ± 12	100	61 ± 42
		100	10 ± 2	100	10 ± 2	100	11 ± 3	100	27 ± 30
7	✓	100	14 ± 3	100	13 ± 3	100	27 ± 13	100	84 ± 51
		100	13 ± 2	100	12 ± 2	100	14 ± 3	100	34 ± 37
8	✓	100	16 ± 3	100	16 ± 3	100	32 ± 15	100	104 ± 60
		100	15 ± 3	100	15 ± 3	100	17 ± 4	100	40 ± 44
9	✓	100	19 ± 4	100	19 ± 4	100	38 ± 17	100	132 ± 71
		100	19 ± 3	100	18 ± 3	100	20 ± 4	100	48 ± 52
10	✓	100	21 ± 4	100	22 ± 4	100	44 ± 20	100	164 ± 85
		100	22 ± 4	100	21 ± 3	100	23 ± 5	100	57 ± 64
11	✓	100	24 ± 4	100	26 ± 7	100	50 ± 23	100	201 ± 101
		100	25 ± 5	100	24 ± 4	100	26 ± 5	100	66 ± 74
12	✓	100	27 ± 5	100	30 ± 8	100	56 ± 27	100	243 ± 117
		100	29 ± 7	100	28 ± 4	100	30 ± 5	100	77 ± 91
13	✓	100	30 ± 5	100	34 ± 8	100	64 ± 32	100	290 ± 136
		100	33 ± 9	100	31 ± 4	100	33 ± 5	100	92 ± 112
14	✓	100	34 ± 6	100	38 ± 9	100	73 ± 38	100	346 ± 153
		100	38 ± 12	100	34 ± 4	100	36 ± 6	100	109 ± 138
15	✓	100	37 ± 6	100	42 ± 9	100	82 ± 45	99	403 ± 168
		100	43 ± 15	100	37 ± 5	100	39 ± 6	99	129 ± 167
16	✓	100	40 ± 6	100	47 ± 10	100	95 ± 55	94	463 ± 176
		100	49 ± 19	100	41 ± 5	100	43 ± 7	96	144 ± 189
17	✓	100	43 ± 7	100	50 ± 11	100	106 ± 65	88	494 ± 175
		99	52 ± 21	100	44 ± 5	100	46 ± 8	93	145 ± 189
18	✓	100	46 ± 7	100	54 ± 12	100	121 ± 81	78	533 ± 173
		97	57 ± 22	100	47 ± 6	100	50 ± 9	90	144 ± 185
19	✓	100	50 ± 8	100	58 ± 14	99	136 ± 94	65	561 ± 171
		95	61 ± 23	100	51 ± 6	100	54 ± 10	87	145 ± 180
20	✓	100	55 ± 9	100	65 ± 16	97	158 ± 107	41	593 ± 171
		88	67 ± 23	100	56 ± 6	100	59 ± 12	83	148 ± 175

Table 14 Success rate, as well as average and standard deviation of the number of qubits required by the minor embedding built by the RL agent trained on the randomly generated problem graphs G up to 30 nodes (shown from$|G| = 21$to 30 on the rows) mapped to Zephyr topology graphs H (with$H_{\text{size}} \in [2, 5]$on the columns). The DA column distinguishes between the agents trained without data augmentations and the ones where data augmentations are used both during training and testing (✓)

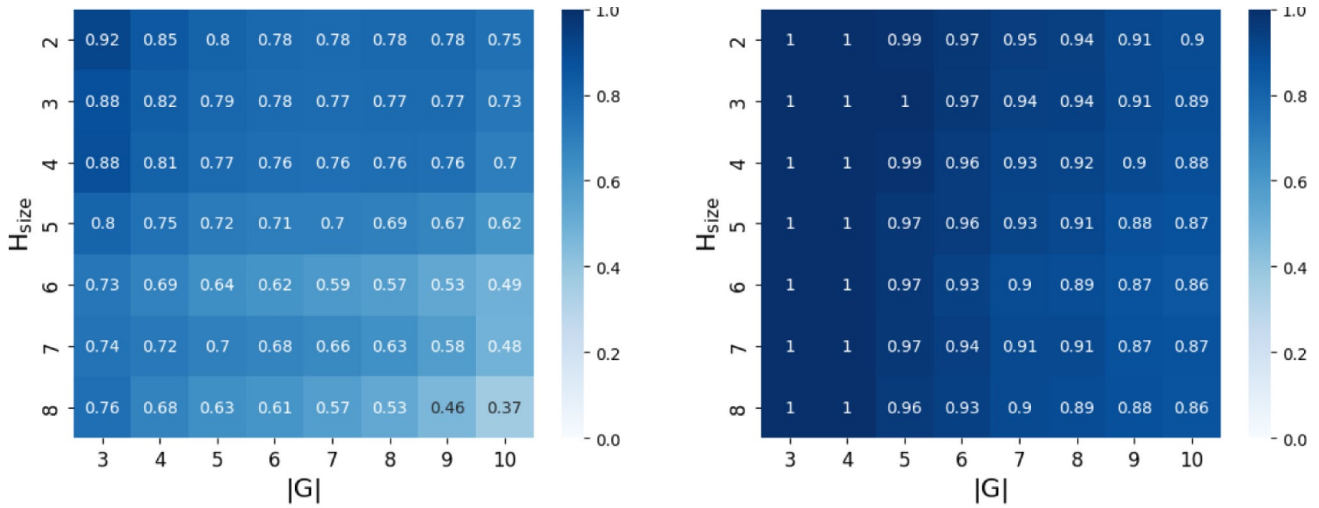
$\|\mathbf{G}\|$	**DA**	$\mathbf{H_{size} = 2}$		$\mathbf{H_{size} = 3}$		$\mathbf{H_{size} = 4}$		$\mathbf{H_{size} = 5}$	
		SR%	#Q	SR%	#Q	SR%	#Q	SR%	#Q
21	✓	100	58 ± 10	100	68 ± 17	95	162 ± 108	37	579 ± 166
		87	68 ± 22	100	59 ± 7	100	63 ± 14	82	154 ± 177
22	✓	99	63 ± 12	100	74 ± 20	92	173 ± 112	28	579 ± 165
		82	72 ± 21	100	63 ± 8	100	68 ± 18	80	160 ± 179
23	✓	99	68 ± 13	100	80 ± 22	88	181 ± 112	21	587 ± 155
		78	77 ± 21	100	68 ± 9	100	75 ± 22	77	165 ± 176
24	✓	98	74 ± 15	100	86 ± 24	83	184 ± 107	17	593 ± 151
		72	81 ± 22	100	73 ± 11	100	82 ± 29	75	175 ± 178
25	✓	95	81 ± 17	99	95 ± 27	78	189 ± 102	13	607 ± 142
		65	87 ± 23	100	79 ± 15	100	92 ± 37	72	184 ± 177
26	✓	90	87 ± 18	98	103 ± 30	73	197 ± 99	10	612 ± 137
		56	91 ± 24	100	87 ± 20	100	105 ± 50	68	195 ± 178
27	✓	82	93 ± 19	97	112 ± 33	69	212 ± 100	8	625 ± 128
		46	95 ± 24	99	97 ± 28	99	122 ± 66	64	206 ± 178
28	✓	71	99 ± 20	95	121 ± 36	64	229 ± 103	6	637 ± 122
		37	98 ± 24	97	108 ± 37	96	141 ± 80	59	221 ± 181
29	✓	57	103 ± 20	92	130 ± 41	58	244 ± 106	5	649 ± 118
		29	101 ± 24	93	120 ± 44	91	161 ± 93	54	237 ± 186
30	✓	44	108 ± 20	85	142 ± 47	50	262 ± 108	3	668 ± 108
		23	105 ± 24	85	131 ± 50	83	179 ± 103	48	253 ± 189

B.1 QER results for the randomly generated graphs



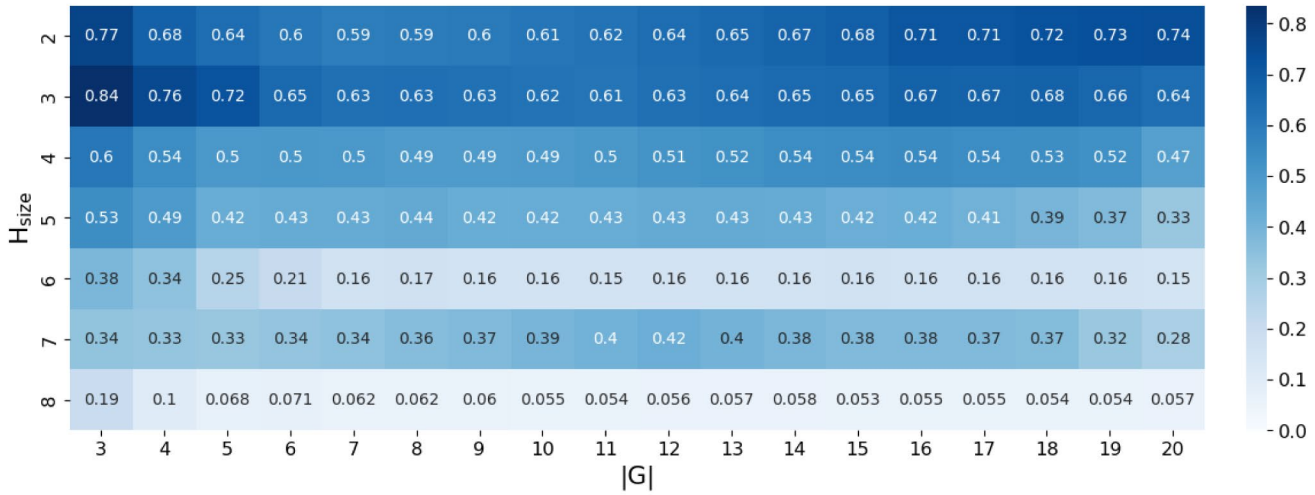
(a) Average QER (b) Best QER

Fig. 17 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 10$

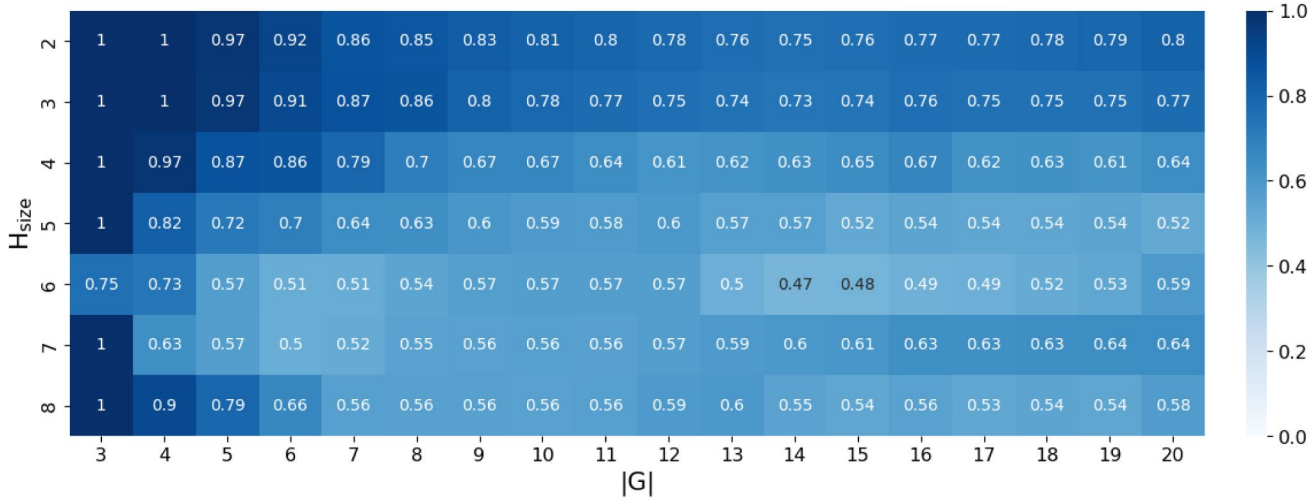


(a) Average QER (b) Best QER

Fig. 18 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 10$ using data augmentation both in training and testing

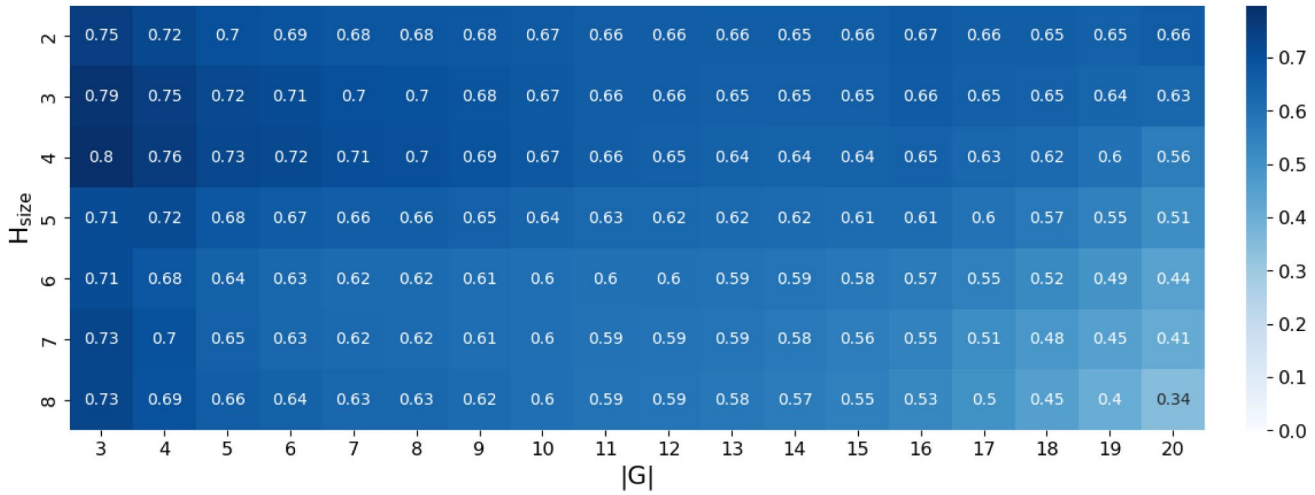


(a) Average QER

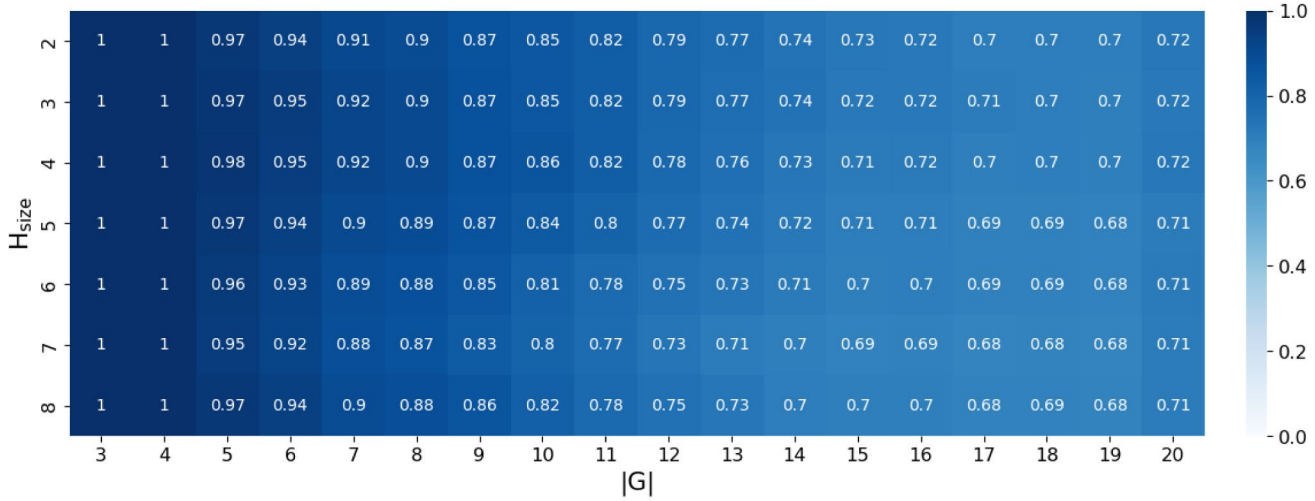


(b) Best QER

Fig. 19 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 20$

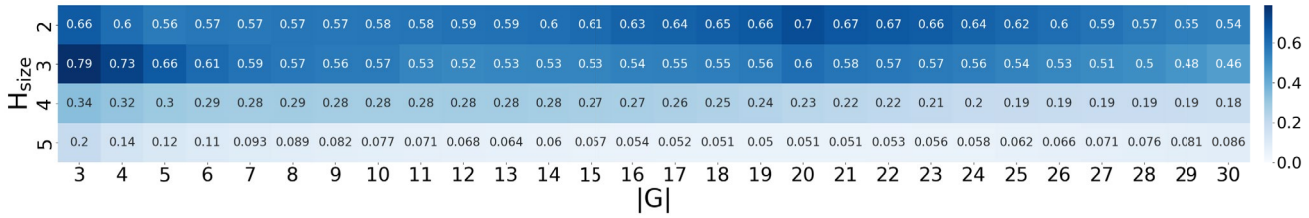


(a) Average QER

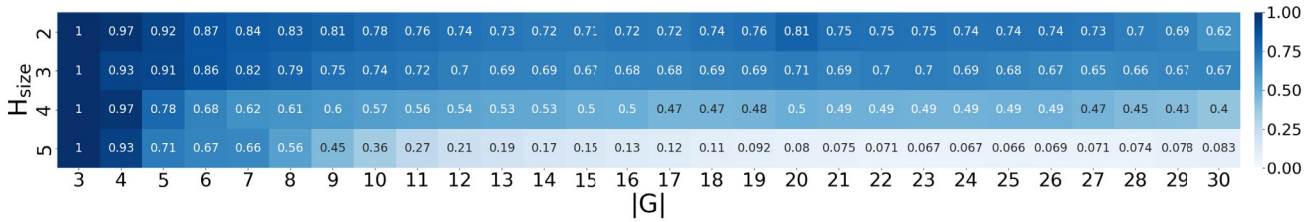


(b) Best QER

Fig. 20 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 20$ using data augmentation both in training and testing

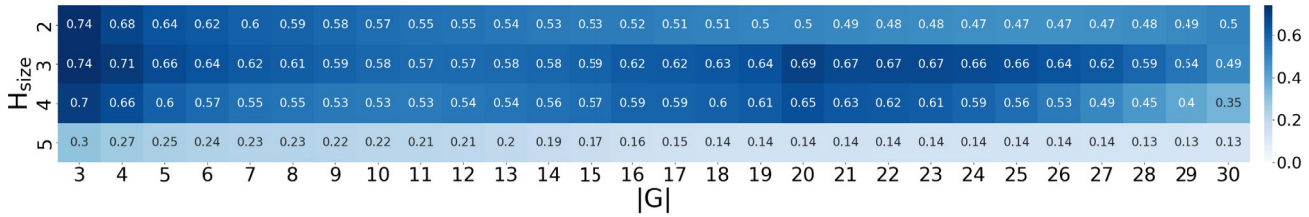


(a) Average QER

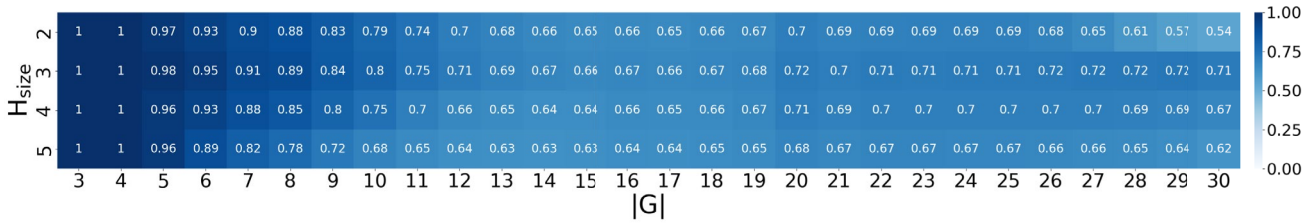


(b) Best QER

Fig. 21 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 30$



(a) Average QER



(b) Best QER

Fig. 22 Qubit Efficiency Ratio of the RL agent on the Zephyr topology when trained on random graphs up to $|G| = 30$ using data augmentation both in training and testing

Acknowledgements We acknowledge the financial support from ICSC - "National Research Centre in High Performance Computing, Big Data and Quantum Computing", funded by European Union - NextGenerationEU.

Author Contributions R. Nembrini conceived the original idea. R. Nembrini and M. Ferrari Dacrema designed the experimental methodology and R. Nembrini carried out the experiments. The main manuscript text was written by R. Nembrini and M. Ferrari Dacrema and all authors contributed to the final version. All authors reviewed and approved the final manuscript.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

Data Availability Research data, consisting in a dataset of randomly generated graphs, can be found at the following URL: https://github.com/qcpolimi/RLxME_Dataset

Declarations

Competing interests The authors declare no competing interests.

References

Bernal DE, Booth KE, Dridi R, et al (2020) Integer programming techniques for minor-embedding in quantum annealers. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21–24, 2020, Proceedings 17, Springer, pp 112–129

Boothby T, King AD, Roy A (2016) Fast clique minor generation in chimera qubit connectivity graphs. Quantum Inf Process 15(1):495–508. https://doi.org/10.1007/s11128-015-1150-6

Born M, Fock V (1928) Beweis des adiabatensatzes Zeitschrift für Physik 51(3):165–180. https://doi.org/10.1007/BF01343193

Cai J, Macready WG, Roy A (2014) A practical heuristic for finding graph minors. CoRR abs/1406.2741. http://arxiv.org/abs/1406.2741

Carugno C, Ferrari Dacrema M, Cremonesi P (2022) Evaluating the job shop scheduling problem on a d-wave quantum annealer. Nat Sci Rep 12(1):6539–655. https://doi.org/10.1038/s41598-022-10169-0

Carugno C, Ferrari Dacrema M, Cremonesi P (2024) Adaptive learning for quantum linear regression. In: Osinski M, Cour BL, Yeh L (eds) IEEE International Conference on Quantum Computing and Engineering, QCE 2024, Montreal, QC, Canada, September 15-20, 2024. IEEE, pp 1595–1599,https://doi.org/10.1109/QCE60285.2024.00186

Chiavassa P, Marchesin A, Pedone I, et al (2022) Virtual network function embedding with quantum annealing. In: (ed) IEEE International Conference on Quantum Computing and Engineering, QCE 2022, Broomfield, CO, USA, September 18-23, 2022. IEEE, pp 282–291,https://doi.org/10.1109/QCE53715.2022.00048

Fang YL, Warburton P (2020) Minimizing minor embedding energy: an application in quantum annealing. Quantum Inf Process 19(7):191

Farhi E, Goldstone J, Gutmann S, et al (2000) Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106

Ferrari Dacrema M, Moroni F, Nembrini R, et al (2022) Towards feature selection for ranking and classification exploiting quantum annealers. In: Amigó E, Castells P, Gonzalo J, et al (eds) SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022. ACM, pp 2814–2824, https://doi.org/10.1145/3477495.3531755

Glover F, Kochenberger G, Hennig R et al (2022) Quantum bridge analytics i: a tutorial on formulating and using qubo models. Ann Oper Res 314:141–183. https://doi.org/10.1007/s10479-022-04634-2

Grondman I, Busoniu L, Lopes GAD et al (2012) A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Trans Syst Man Cybern Part C 42(6):1291–1307. https://doi.org/10.1109/TSMCC.2012.2218595

Hernandez M, Aramon M (2017) Enhancing quantum annealing performance for the molecular similarity problem. Quantum Inf Process 16(5):133. https://doi.org/10.1007/s11128-017-1586-y

Huang S, Ontañón S (2022) A closer look at invalid action masking in policy gradient algorithms. In: Barták R, Keshtkar F, Franklin M (eds) Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022, https://doi.org/10.32473/flairs.v35i.130584

Ikeda K, Nakamura Y, Humble TS (2019) Application of quantum annealing to nurse scheduling problem. Sci Rep 9:12837. https://doi.org/10.1038/s41598-019-49172-3

Johnson MW, Amin MH, Gildert S et al (2011) Quantum annealing with manufactured spins. Nature 473(7346):194–198

Kadowaki T, Nishimori H (1998) Quantum annealing in the transverse ising model. Phys Rev E 58:5355–5363. https://doi.org/10.1103/PhysRevE.58.5355

Konda VR, Tsitsiklis JN (1999) Actor-critic algorithms. In: Solla SA, Leen TK, Müller K (eds) Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]. The MIT Press, pp 1008–1014, http://papers.nips.cc/paper/1786-actor-critic-algorithms

Kumar V, Bass G, Tomlin C et al (2018) Quantum annealing for combinatorial clustering. Quantum Inf Process 17(2):39. https://doi.org/10.1007/s11128-017-1809-2

Lucas A (2014) Ising formulations of many np problems. Frontiers in physics 2:5

Mandrá S, Zhu Z, Wang W et al (2016) Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics versus quantum approaches. Phys Rev A 94:022337. https://doi.org/10.1103/PhysRevA.94.022337

Micheletti C, Hauke P, Faccioli P (2021) Polymer physics by quantum computing. Phys Rev Lett 127:080501. https://doi.org/10.1103/PhysRevLett.127.080501

Mnih V, Badia AP, Mirza M, et al (2016) Asynchronous methods for deep reinforcement learning. In: Balcan M, Weinberger KQ (eds) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, JMLR Workshop and Conference Proceedings, vol 48. JMLR.org, pp 1928–1937, http://proceedings.mlr.press/v48/mniha16.html

Mott A, Job J, Vlimant JR et al (2017) Solving a higgs optimization problem with quantum annealing for machine learning. Nature 550:375–379. https://doi.org/10.1038/nature24047

Nembrini R, Ferrari Dacrema M, Cremonesi P (2021) Feature selection for recommender systems with quantum computing. Entropy 23(8):970. https://doi.org/10.3390/e23080970

Nembrini R, Carugno C, Ferrari Dacrema M, et al (2022) Towards recommender systems with community detection and quantum computing. In: Golbeck J, Harper FM, Murdock V, et al (eds) RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022. ACM, pp 579–585, https://doi.org/10.1145/3523227.3551478

Nembrini R, Ferrari Dacrema M, Cremonesi P (2024) An application of reinforcement learning for minor embedding in quantum annealing. In: Baioletti M, González MÁ, Oddi A, et al (eds) Proceedings of the International Workshop on AI for Quantum and Quantum for AI (AIQxQIA 2024) co-located with 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), November 25 - November 28, 2024, Free University of Bolzano, Bolzano, Italy, CEUR Workshop Proceedings, vol 3913. CEUR-WS.org, https://ceur-ws.org/Vol-3913/short2.pdf

Neukart F, Dollen DV, Seidel C (2018) Quantum-assisted cluster analysis on a quantum annealing device. Frontiers in Physics 6:55. https://doi.org/10.3389/fphy.2018.00055

Neukart F, Von Dollen D, Seidel C et al (2018) Quantum-enhanced reinforcement learning for finite-episode games with discrete state spaces. Frontiers in Physics 5:71. https://doi.org/10.3389/fphy.2017.00071

Neven H, Denchev VS, Rose G, et al (2009) Training a large scale classifier with the quantum adiabatic algorithm. CoRR abs/0912.0779. http://arxiv.org/abs/0912.0779

Ngo H, Do N, Vu M et al (2025) Charme: A chain-based reinforcement learning approach for the minor embedding problem. ACM Transactions on Quantum Computing 7(1):1–28. https://doi.org/10.1145/3763244

Ngo HM, Kahveci T, Thai MT (2023) ATOM: an efficient topology adaptive algorithm for minor embedding in quantum computing. In: IEEE International Conference on Communications, ICC 2023, Rome, Italy, May 28 - June 1, 2023. IEEE, pp 2692–2697. https://doi.org/10.1109/ICC45041.2023.10279010

Ohzeki M (2020) Breaking limitation of quantum annealer in solving optimization problems under constraints. CoRR abs/2002.05298. https://arxiv.org/abs/2002.05298

O'Malley D, Vesselinov VV, Alexandrov BS, et al (2017) Nonnegative/binary matrix factorization with a d-wave quantum annealer. CoRR abs/1704.01605. http://arxiv.org/abs/1704.01605,

Ottaviani D, Amendola A (2018) Low rank non-negative matrix factorization with d-wave 2000q. https://doi.org/10.48550/arXiv.1808.08721, [quant-ph]

Pasin A, Ferrari Dacrema M, Cremonesi P, et al (2024) Overview of QuantumCLEF 2024: The quantum computing challenge for information retrieval and recommender systems at CLEF. In: Goeuriot L, Mulhem P, Quénot G, et al (eds) Experimental IR Meets Multilinguality, Multimodality, and Interaction - 15th International Conference of the CLEF Association, CLEF 2024, Grenoble, France, September 9-12, 2024, Proceedings, Part II, Lecture Notes in Computer Science, vol 14959. Springer, pp 260–282, https://doi.org/10.1007/978-3-031-71908-0_12

Pellini R, Ferrari Dacrema M (2024) Analyzing the effectiveness of quantum annealing with meta-learning. Quantum Machine Intelligence 6(2):48. https://doi.org/10.1007/S42484-024-00179-8

Pelofske E (2024) 4-clique network minor embedding for quantum annealers. Phys Rev Appl 21(3):034023

Raffin A, Hill A, Gleave A, et al (2021) Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research 22(268):1–8. http://jmlr.org/papers/v22/20-1364.html

Rieffel EG, Venturelli D, O'Gorman B et al (2015) A case study in programming a quantum annealer for hard operational planning problems. Quantum Inf Process 14(1):1–36. https://doi.org/10.1007/s11128-014-0892-x

Schulman J, Levine S, Abbeel P, et al (2015) Trust region policy optimization. In: Bach FR, Blei DM (eds) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, JMLR Workshop and Conference Proceedings, vol 37. JMLR.org, pp 1889–1897, http://proceedings.mlr.press/v37/schulman15.html

Schulman J, Wolski F, Dhariwal P, et al (2017) Proximal policy optimization algorithms. CoRR abs/1707.06347. http://arxiv.org/abs/1707.06347,

Shorten C, Khoshgoftaar TM (2019) A survey on image data augmentation for deep learning. J Big Data 6:60. https://doi.org/10.1186/s40537-019-0197-0

Silver D, Huang A, Maddison CJ et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489. https://doi.org/10.1038/nature16961

Silver D, Hubert T, Schrittwieser J et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362(6419):1140–1144. https://doi.org/10.1126/science.aar6404

Stollenwerk T, Lobe E, Jung M (2017) Flight gate assignment with a quantum annealer. In: Feld S, Linnhoff-Popien C (eds) Quantum Technology and Optimization Problems - First International Workshop, QTOP@NetSys 2019, Munich, Germany, March 18, 2019, Proceedings, Lecture Notes in Computer Science, vol 11413. Springer, pp 99–110, https://doi.org/10.1007/978-3-030-14082-3_9

Streif M, Neukart F, Leib M (2019) Solving quantum chemistry problems with a d-wave quantum annealer. https://doi.org/10.48550/arXiv.1811.05256, arXiv:1811.05256, [quant-ph]

Sutton RS, Barto AG (2018) Reinforcement learning - an introduction, 2nd Edition. MIT Press, http://www.incompleteideas.net/book/the-book-2nd.html

Towers M, Kwiatkowski A, Terry J, et al (2024) Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint arXiv:2407.17032

Vinci W, Albash T, Paz-Silva G et al (2015) Quantum annealing correction with minor embedding. Phys Rev A 92(4):042310

Willsch D, Willsch M, Raedt HD et al (2020) Support vector machines on the d-wave quantum annealer. Comput Phys Commun 248:107006. https://doi.org/10.1016/j.cpc.2019.107006

Xia R, Bian T, Kais S (2018) Electronic structure calculations and the ising hamiltonian. J Phys Chem B 122(13):3384–3395. https://doi.org/10.1021/acs.jpcb.7b10371