DeepCubeAF: A Foundation Model for Generalizable Pathfinding Heuristics

Anonymous authors
Paper under double-blind review

Keywords: foundation models, generalization in planning, deep reinforcement learning, pathfinding

Summary

We present DeepCubeAF, a method for training a foundation model for heuristic functions that generalize across pathfinding domains. Our approach uses a domain generator to generate training data, trains a graph neural network (GNN) using deep reinforcement learning and hindsight experience replay, and uses batch weighted A* search to solve problems. Our experiments show that DeepCubeAF consistently solves more problems than foundation models based on supervised learning, is competitive with traditional domain-independent planners, and can solve problems from domains not seen during training.

Contribution(s)

1. We introduce DeepCubeAF, a foundation model trained with reinforcement learning to learn heuristic functions for solving pathfinding problems.
   Context: Previous methods rely on supervised learning and, therefore, assume training instances can already be solved. Traditional domain-independent methods do not make use of machine learning, and thus must rely only on hand-designed heuristics.
2. We demonstrate that the resulting model can generalize to previously unseen domains by applying without additional fine-tuning.
   Context: This shows that deep reinforcement learning is a viable approach for creating a domain-independent solver. This will allow practitioners that are not familiar with machine learning algorithms to solve pathfinding problems with machine learning models.

DeepCubeAF: A Foundation Model for Generalizable Pathfinding Heuristics

Anonymous authors
Paper under double-blind review

Abstract

Pathfinding problems can be found in fields such as robotics, mathematics, chemistry, and program synthesis, where the objective of pathfinding is to find a sequence of actions that transforms a given start state into a goal state. Recently, deep reinforcement learning (DRL) has emerged as a promising method for automatically training domain-specific heuristic functions to solve these problems in a largely domain-independent fashion. However, these approaches often require retraining for even a slight change in domain, resulting in significant resource and time inefficiencies. While existing approaches use supervised learning to learn generalizable heuristics to handle unseen domains, they are limited by the need to obtain supervised labels. We draw inspiration from domain randomization in reinforcement learning to handle these limitations and the DeepCubeA algorithm and introduce DeepCubeA for foundation models (DeepCubeAF). DeepCubeAF trains a heuristic function across randomly generated domains using reinforcement learning and uses this trained heuristic function with batch weighted A* search to solve problems. Our model consistently shows better generalizability than the existing foundation model for both seen and unseen domains. This work represents a step toward training robust, generalizable models and providing access to these models to experts across various fields.

1 Introduction

Pathfinding aims to find a sequence of actions that forms a path from a given start state to a given goal state while minimizing the total path cost. Pathfinding problems, many of which are also studied in reinforcement learning, are prevalent across computer science, robotics, mathematics, and the natural sciences. Heuristic search, one of the most prominent approaches to solving pathfinding problems, relies on a heuristic function which estimates the "cost-to-go," which is the cost of the shortest path from a given state to a nearest goal state. Recently, deep reinforcement learning (DRL) (Sutton & Barto, 2018) based methods have successfully been used to learn domain-specific heuristics in a largely domain-independent fashion to solve pathfinding problems such as puzzle solving (Agostinelli et al., 2019; 2024), chemical synthesis (Chen et al., 2020), and quantum algorithm compilation (Zhang et al., 2020). However, training such heuristic functions using deep neural networks (DNNs) (Schmidhuber, 2015) can take days and requires substantial computational resources, often necessitating retraining for even minor domain changes. These hardware and time constraints limit accessibility of these methods for researchers not familiar with machine learning algorithms and hardware and present a bottleneck when attempting to solve problems across many different domains.

These challenges with hardware and time constraints when using DRL to solve pathfinding problems are common across applications of machine learning, such as those encountered in computer vision and natural language processing. To address these challenges, foundation models have been developed, which are large DNNs trained on large, diverse, datasets that can then be adapted to new

tasks with minimal to no fine-tuning (Szegedy et al., 2015; Devlin et al., 2019; Radford et al., 2019). Based on the success of foundation models for other applications of machine learning, we will create a foundation model for heuristic functions for pathfinding problems.

To create our foundation model, we will use the planning domain description language (PDDL) (McDermott, 2000), which is a description language commonly used by the planning community to represent pathfinding problems. A PDDL description of the domain, along with a PDDL description of the problem instance, which consists of a start state and a goal, will be given to a graph neural network (GNN), which represents the heuristic function. To generate training data, we will build on PDDLFUSE (Khandelwal et al., 2024) to generate domains and hindsight experience replay (HER) (Andrychowicz et al., 2017) to generate training problem instances. To train the heuristic function, we will build on the DeepCubeA algorithm (Agostinelli et al., 2019) by using approximate value iteration (AVI) (Bertsekas & Tsitsiklis, 1996). To solve problem instances in a given domain, we will also build on DeepCubeA by using batch weighted A* search (Hart et al., 1968; Pohl, 1970) with the trained heuristic function. We call this algorithm DeepCubeA for foundation models (DeepCubeAF). Our experiments show that DeepCubeAF can generalize to domains not seen during training and outperforms foundation models trained only using supervised learning.

2 Related Work

2.1 Generalization in Pathfinding Problems

Domain-independent planners, such as the fast downward planner (Helmert, 2006), rely on the automated construction of heuristic functions based on a PDDL description of a domain. These methods include the fast forward (Hoffmann & Nebel, 2001), causal graph, and goal count heuristics. However, experiments have shown that these methods do not perform as well as learned heuristic functions (Agostinelli et al., 2024). Large language models (LLMs), have been used to solve pathfinding via in-context learning (Sermanet et al., 2023; Li et al., 2023; Silver et al., 2023); however, they face notable challenges that include syntax errors and lack inherent search capabilities. Although, future work using chain-of-thought reasoning may be able to overcome this.

Chen et al. (2024) introduced three novel graph representations for planning tasks using GNNs to learn domain-independent heuristics. Their approach mitigates issues with large grounded GNNs by leveraging lifted representations and demonstrates superior generalization to larger problems compared to models like STRIPS-HGN (a hypergraph network model that learns domain-independent planning heuristics directly from the delete-relaxed representation of STRIPS problems using a recurrent encode-process-decode architecture)(Shen et al., 2020). However, it faces scalability issues with large graph construction. Building on this, Chen et al. (2024) proposed the GOOSE framework using GNNs with novel grounded and lifted graph representations for classical planning to learn a generalized heuristic function with supervised learning. Their heuristics outperform STRIPS-HGN and hFF (the Fast Forward heuristic (Hoffmann & Nebel, 2001), which computes cost-to-go estimates by generating relaxed plans that ignore delete effects) in various domains, however, since they use supervised learning, their approach assumes that ability to find optimal solutions for all training instances. However, this is not possible for many pathfinding tasks without significant domain-dependent effort (Agostinelli et al., 2024). Additionally, (Toyer et al., 2018) utilized GNNs to improve coverage and plan quality in classical planning tasks through new graph representations, though their approach only generalizes across a subset of test domains.

2.2 GOOSE

Our work also compares with an existing generalizable method for solving planning domains. (Chen et al., 2024) uses the STRIPS Learning Graph (SLG) representation, which encodes states, actions, and their relationships into a structured graph. SLG captures domain knowledge by representing preconditions and effects and deleting effects of actions as graph edges, enabling the model to learn heuristics through Graph Neural Networks (GNNs).

This approach combines domain-specific training with domain-independent methods, leveraging the SLG's ability to generalize within a domain. The GNN propagates information through the SLG to compute heuristic values, allowing the model to handle larger instances of the same domain without retraining.

Although primarily domain-specific, the SLG-based model aligns with foundational principles by utilizing structural representations and scalable learning techniques. It demonstrates how graph-based representations can improve adaptability in classical planning tasks, particularly for domains like n-puzzle.

2.3 Domain Randomization in Reinforcement Learning

Domain randomization has proven highly effective in reinforcement learning (RL) for training agents that generalize to unseen environments. By training on multiple randomized variants of a domain, RL agents learn to handle discrepancies in states, transitions, or dynamics. For instance, Mehta et al. (2020) introduce Active Domain Randomization, which adaptively selects challenging environments for training, while Ajani et al. (2023) demonstrate improved robustness by randomizing physical parameters such as friction in robotics. These successes highlight the power of exposing learning systems to diverse environments, a principle that can be translated to automated planning domains.

3 Preliminaries

3.1 Foundation Models

Foundation models are deep learning models pre-trained on extensive, diverse datasets. Once trained, F_θ is expected to generalize across different domains and tasks with or without fine-tuning. Foundation models have significant applications in computer vision Szegedy et al. (2015) and natural language processing (NLP) Devlin et al. (2019); Radford et al. (2019).

Definition: A foundation model F_θ is trained to minimize the loss function L over a dataset D:

$$\theta^* = \arg\min_\theta L(F_\theta(x), x) \quad \text{for} \quad x \in D$$

3.2 Pathfinding

A pathfinding domain can be represented by a weighted directed graphs where nodes represent states, edges represent transitions between states (actions), and weights represent transition costs. Given a domain, a pathfinding problem instance is defined by a start state and a set of goal states. A solution to a pathfinding problem is a sequence of actions that forms a path between the start state and a goal state, with preference for solutions with cheaper path costs, where the path cost is the sum of transition costs along a given path. Traditionally, pathfinding employs heuristic search algorithms, which make use of a heuristic function that maps states to an estimate of the cost-to-go.

Approximate Value Iteration Given a tabular representation for the heuristic function, value iteration can be used to find h^*, which maps states to the cost of a shortest path using the following equation as an update rule:

$$h'(s) = \min_{a \in \mathcal{A}} \left(c(s,a) + h(T(s,a)) \right) \tag{1}$$

where $h(s)$ represents the current estimate of the cost-to-go, $h'(s)$ is the updated estimate, $c(s,a)$ is the cost of taking action a in state s, and $T(s,a)$ is the resulting state after applying action a. Since pathfinding problems are deterministic, value iteration can use a deterministic state transition function instead of transition probabilities.

Tabular methods become impractical for domains with large state spaces, which are common in real-world problems. To address this, we use approximate value iteration (AVI), where a parameterized function approximates value iteration updates. In AVI, a DNN with parameters θ is used to represent the heuristic function. The network refines its estimates for the cost-to-go values by minimizing the loss:

$$L(\theta) = \left(\min_{a \in \mathcal{A}} \left(c(s,a) + h_{\theta^-}(T(s,a)) \right) - h_\theta(s) \right)^2 \tag{2}$$

where $h_\theta(s)$ approximates the cost-to-go and θ^- denotes the parameters of a target network (Mnih et al., 2015), which is periodically updated to θ.

3.3 Batch Weighted A* Search

A* search (Hart et al., 1968) is a widely used algorithm for solving pathfinding problems. A* search maintains a search tree, where nodes represent states and edges represent transitions between states. Nodes are expanded according to a priority, f, shown in Equation 3:

$$f(n) = g(n) + h(n) \tag{3}$$

where $g(n)$ is the path cost from the starting node to node n and $h(n)$ is the cost-to-go estimate from the state associated with node n to a closest goal state provided by a heuristic function. The algorithm terminates when a node associated with a goal state is selected for expansion.

In order to trade potentially faster search times with potentially longer path costs a weight, λ, can be used to reduce the contribution of, g, to the total cost, as shown in Equation 4, where $0 \leq \lambda \leq 1$. Furthermore, when the heuristic function is represented by a DNN, parallelism provided by graphics processing units (GPUs) can be exploited by expanding N nodes at a time instead of just one using batch weighted A* search (BWAS) (Agostinelli et al., 2019; Li et al., 2022).

$$f(n) = \lambda g(n) + h(n) \tag{4}$$

3.4 Planning Domain Definition Language (PDDL)

The Planning Domain Definition Language (PDDL) provides a formal framework for representing planning problems, where an agent transitions through states by executing actions to achieve a goal. A PDDL problem consists of a *domain* $\mathcal{D}$ and a *problem* $\mathcal{P}$. The domain $\mathcal{D}$ defines the set of possible actions and how they effect a given state, while the problem $\mathcal{P}$ specifies an initial state s_0 and a goal condition G.

A PDDL domain is defined by:

- **Objects** $\mathcal{O}$: The entities in the domain.
- **Predicates** $\mathcal{P}$: Boolean functions describing state conditions.
- **Actions** $\mathcal{A}$: Each action $a \in \mathcal{A}$ is defined by:
  - Parameters: A set of objects required for execution.
  - Preconditions $\text{Pre}(a) \subseteq \mathcal{P}$: Conditions that must hold before execution.
  - Effects $\text{Eff}(a) \subseteq \mathcal{P}$: State changes resulting from execution.

A problem instance $\mathcal{P}$ consists of an initial state $s_0 \subseteq \mathcal{P}$ and a goal condition $G \subseteq \mathcal{P}$. The objective is to find a sequence of actions $\pi = (a_1, a_2, \ldots, a_n)$ such that executing π from s_0 results in a state satisfying G.

3.5 PDDLFuse: Generating Diverse Planning Domains

Taking inspiration from domain randomization techniques in reinforcement learning (Mehta et al., 2020; Ajani et al., 2023), PDDLFuse (Khandelwal et al., 2024) generates planning domains in

PDDL format by randomly modifying and combining (fusing) existing domains. PDDLFUSE does this by by: 1) merging predicate sets and action schemas from multiple domains; 2) randomly adding or removing preconditions and/or effects; 3) introducing controlled negation and reversibility flags for selected predicates. Empirical results show that standard domain-independent planners often struggle to solve the more challenging PDDLFUSE domains.

4 Methodology

4.1 Enhanced Domain and Problem Instance Generation

Expanded Base Domains and Problem Files. Our modifications to PDDLFUSE begin by significantly increasing the set of base domains and problem files. This expansion ensures more diverse set of initial problem files per generated domains. When fusing domains, we preserve predicate and action names by applying systematic renaming, ensuring there is no overlap among symbols across different domains. The final fused domain thus contains a merged set of predicates and actions with randomized modifications to preconditions and effects.

Adaptive Negation. A key step in domain fusion involves adding or removing preconditions and effects from actions. We introduce an adaptive negation mechanism that selectively identifies predicates eligible for negation in the newly fused domain's problem file. Specifically, a predicate is deemed eligible if it is not present in the critical preconditions of actions—since negating such predicates could prevent actions from being executed and render the problem unsolvable. By applying negation with a user-controlled probability only to these non-essential predicates, our approach preserves the core functionality of the domain while still introducing useful variability. By carefully targeting the negated predicates, we improve the consistency of generated actions and the solvability of resultant problems.

4.2 Revised Action Generation Technique

Previously, PDDLFUSE created problem files by randomly selecting actions from the fused domain and subsequently pruning to form a valid plan. This random approach was often inefficient, requiring substantial computation to verify solvability and identify relevant actions. We replace it with a more planner-aligned technique designed to efficiently produce action sequences and solvable planning problems.

Planner-Aligned Sampling. Instead of choosing actions uniformly at random, we adopt a technique akin to a forward search used in classical planners. We begin with an initial state derived from the domain objects and predicates, then dynamically select only the set of applicable actions—those whose preconditions are satisfied in the current state. Among these applicable actions, we apply a stochastic selection (e.g., uniform or heuristic-based) to pick one action. We then apply this action to reach a successor state.

Controlled Sequence Generation. We repeat the above process for a user-configurable number of steps (or until no applicable actions remain). This controlled, state-based generation ensures that we continuously build a coherent action sequence, thereby avoiding many dead-end paths. Once the final state is reached, we designate a subset of its predicates as the *goal* conditions. The resulting domain and problem pair therefore includes a sequence of actions known to transition the initial state to the goal state. This process is summarized in Algorithm 1. This approach is similar to that of hindsight experience replay (HER) Andrychowicz et al. (2017) in that the state at the end of a sequence of actions is used to create a goal.

Computational Benefits. Aligning the action generator with planner logic significantly lowers the computational overhead. Instead of repeatedly checking for plan validity after random selections, the state-based approach validates each action as it is chosen, mitigating expensive re-planning or

Algorithm 1 Revised Action Generator

Require: Fused domain $\mathcal{D}$, maxSteps M
state $\leftarrow$ randomlyInstantiate($\mathcal{D}$)
for $i = 1$ to M **do**
  applicableActions $\leftarrow \{a \in \mathcal{A} \mid \text{pre}(a) \subseteq \text{initializeState}\}$
  if applicableActions $= \varnothing$ **then**
    break ▷ No more valid actions
  end if
  $a^* \leftarrow$ chooseAction(applicableActions)
  state $\leftarrow$ apply(a^*, state)
end for
goalPreds $\leftarrow$ selectSubset(state)
return $(\mathcal{D}, \text{initialState}, \text{goalPreds})$

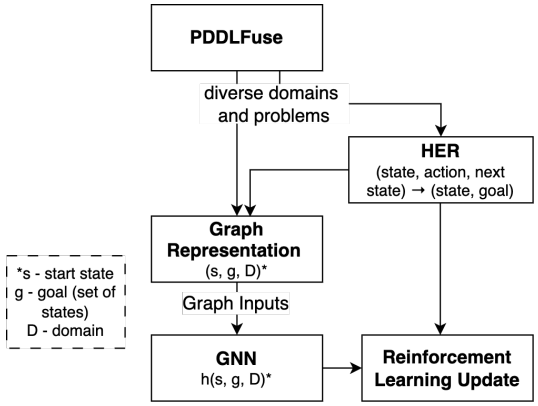

Figure 1: Overview of the training pipeline. PDDLFUSE generates diverse domains and problem instances from which states and goals (which is a set of states) are derived. HER is used to generate problem instances, and a graph-based representation encodes each state, goal (a set of states), and domain as a graph. The GNN maps this representation to an estimate of the cost-to-go.

pruning phases. This streamlined method is especially beneficial when large numbers of domains or problem files are generated, as it accelerates the overall workflow.

4.3 Training the Foundation Model

Figure 1 illustrates our training algorithm which combines PDDLFUSE, HER, GNNs, and reinforcement learning.

State and Goal Generation. We use PDDLFUSE to create diverse planning domains $\mathcal{D}$ and corresponding problem instance Π. Each state, s, is associated with a domain and problem instance. We then sample actions under a given exploration policy to produce transitions (s, a, s'). After each transition, drawing from HER, we create additional goal states, g', by treating s' (or subsets of its predicates) as potential new goals. This process increases the amount of training data by using paths that did not reach the original goal, g.

Graph Representation. Given a state, goal, and domain tuple $(s, g, \mathcal{D})$, we construct a STRIPS Learning Graph (SLG) to encode the planning task (Chen et al., 2024). Each node corresponds to grounded predicates from s, with a feature vector indicating whether the predicate is true in the initial state and whether it is required by the goal g. Edges capture semantic relationships defined by actions: an edge labeled "pre" links an action to a predicate in its preconditions, while edges labeled

"add" or "del" link the action to its positive or negative effects. This concise, structured representation encapsulates the core dependencies between predicates and actions, providing a rich input for our Graph Neural Network to learn effective heuristic functions. This representation produces a set of inputs for a GNN.

GNN Heuristic Function. A GNN, h_θ, takes $(s, g, \mathcal{D})$ as input and outputs an estimate of the cost-to-go, $h_\theta(s, g, \mathcal{D})$. During training, we minimize the value-iteration-based loss shown in Equation 5:

$$L(\theta) = \left(\min_{a \in \mathcal{A}} \left(c(s,a) + h_{\theta^-}(T(s,a), g, \mathcal{D}) \right) - h_\theta(s, g, \mathcal{D}) \right)^2 \tag{5}$$

5 Experiments

We use eight canonical planning domains: Blocks, Ferry, Gripper, N-Puzzle, Sokoban, Spanner, VisitAll, and VisitSome. Each domain has distinct predicates and actions. These domains are used during training for GOOSE and with PDDLFUSE to train DeepCubeAF. A ninth domain, Folding, is used as an additional test domain that was not used during training. We also train a domain-specific heuristic function using DeepCubeA and HER on the folding domain to compare how a heuristic function trained for a specific domain compares to that of DeepCubeAF.

We train DeepCubeAF for one million iterations on domains generated by PDDLFUSE using the Adam optimizer with an initial learning rate of 0.001, with decay of 0.9999993, and batch size of 1000. We encode planning states using a STRIPS Learning Graph (SLG) representation (Chen et al., 2024). This graph captures domain predicates, objects, and actions in a structured manner. All learned models (Vanilla DeepCubeA, DeepCubeAF, and GOOSE) use a message-passing neural network with 16 message-passing layers. This setup follows the GOOSE architecture to maintain consistency among learned models.

We compare to GOOSE, which trains a GNN to represent a heuristic function with supervised learning on the eight aforementioned domains. Since DeepCubeAF is based on reinforcement learning, it does not assume the ability to solve the given problem instances. Therefore, DeepCubeAF can train on many more domains and problem instances. We also compare to the fast downward planner with the fast-forward (FF) heuristic (Helmert, 2006), which is a domain-independent planner that does not use machine learning.

To solve problem instances with the trained heuristic function from DeepCubeAF, we use batch weighted A* search (BWAS) with 100 batch size and 0.8 weight. For GOOSE, we report results available in their paper (Chen et al., 2024) as well as when using the GOOSE heuristic function with our implementation BWAS. Finally, to further test the ability of DeepCubeAF to generalize, we do a leave-one-out experiment, where one of the eight training domains is not used in PDDLFUSE and a heuristic function is trained with data generated from PDDLFUSE with the remaining seven.

6 Results

We evaluate each solver on eight domains plus an additional domain, folding, which is not used for training DeepCubeAF and GOOSE. We use the test instances from the (Chen et al., 2024), as shown in Table 1, which also indicates the parameter ranges for each domain. Table 2 shows results for all solvers. DeepCubeAF achieves higher coverage than GOOSE in all domains, including the leave-one-out (LOO) variants of DeepCubeAF that exclude the test domain. GOOSE_ours, which combines the original GOOSE heuristic function with our implementation of BWAS, solves more instances than GOOSE reported by Chen et al. (2024) (which may be due to the batched node expansions) but still solves fewer problem instances that DeepCubeAF and DeepCubeAF(LOO). DeepCubeAF outperforms DeepCubeAF(LOO), which suggests that additional domain diversity in

training contributes to better generalization. DeepCubeAF solves more problem instances than the fast downward planner in the majority of cases. In the Spanner domain, DeepCubeAF solves 18% of the instances, DeepCubeAF(LOO) solves 7%, whereas FD(FF) and the GOOSE models do not solve any problems.

Neither DeepCubeAF nor GOOSE uses the folding domain during training. DeepCubeA, trained only on folding, attains the highest coverage on folding, as shown in Table 3. Among the domain-independent approaches, DeepCubeAF solves 60% of problem instances while GOOSE and the fast downward planner solve 38%.

domain	**testing**
blocks (90)	$b \in [15, 100]$
ferry (90)	$l, c \in [15, 100]$
gripper (18)	$b \in [15, 100]$
n-puzzle (50)	$n \in [5, 9]$
sokoban (90)	$n \in [8, 12]$
spanner (90)	$s, n \in [15, 100]$
visitall (90)	$n \in [15, 100]$
visitsome (90)	$n \in [15, 100]$
domain-folding (90)	length, folds $\in [15, 100]$

Table 1: Domains, parameter ranges, and number of instances for testing.

7 Discussion

DeepCubeAF uses reinforcement learning to train its heuristic function while GOOSE uses supervised learning. Though supervised learning provides exact labels, reinforcement learning does not assume that the problems it sees can already be solved. In fact, many problems generated by PDDLFUSE cannot be solved with domain-independent planners (Khandelwal et al., 2024), which means they would not be able to be used in a supervised learning setting. However, reinforcement learning will most likely need more examples and longer training times compared to supervised learning since reinforcement learning iteratively updates its own labels. We believe DeepCubeAF was able to outperform GOOSE due to the fact that more diverse training data can be seen during training because it does not assume that labels are given.

One benefit of domain-independent planners that are not based on learning, like the fast-downward planner, is that heuristics can quickly be computed from PDDL files. However, experiments on the folding domain, which was not seen during training for DeepCubeAF, shows that foundation models based on deep reinforcement learning can outperform traditional domain-independent planners. Furthermore, our leave-one-out experiments show DeepCubeAF outperforming the fast-downward planner on three domains. Future research is needed to better define the set of domains for which we can expect foundation mdoels to outperform traditional domain-independent planners.

Future work will investigate how to improve DeepCubeAF, especially for domains not used during training. Foundation models for NLP have showed that larger models and more diverse training data results in better performance. Therefore, one improvement can come from improving the diversity of domains generated by PDDLFUSE. This can possibly be done by leveraging LLMs to generate new PDDL domains instead of just combining existing domains. Another improvement can come from simply training larger models. Finally, given a test domain, limited finetuning can be done. Future work will investigate the effect of finetuning on performance.

Domain	Solver	Len	Nodes	Secs	Nodes/Sec	Solved
	FD(FF)	366.9	4.74E+05	7.7	1.45E+04	74%
	GOOSE	-	-	-	-	10%
blocks	GOOSE_ours	168.25	1.19E+06	65.73	1.81E+04	18%
	DCAF(LOO)	249.15	4.45E+06	228.21	1.95E+04	44%
	DCAF	405.12	1.89E+07	492.14	3.84E+04	**80%**
	FD(FF)	245.47	4.78E+05	2.01	1.29E+04	**95.5%**
	GOOSE	-	-	-	-	31%
ferry	GOOSE_ours	148.75	1.96E+06	86.96	2.27E+04	40%
	DCAF(LOO)	175.09	2.32E+06	148.14	1.56E+04	58%
	DCAF	239.21	4.42E+06	268.68	1.64E+04	92%
	FD(FF)	239	1.67E+04	0.06	6.88E+04	**100%**
	GOOSE	-	-	-	-	28%
gripper	GOOSE_ours	173.13	9.18E+05	58	1.58E+04	44%
	DCAF(LOO)	175.77	1.32E+06	89.2	1.48E+04	50%
	DCAF	220.1	1.51E+06	142.2	1E+04	89%
	FD(FF)	1529.54	1.59E+06	61.84	1.55E+04	70%
	GOOSE	-	-	-	-	12%
n-puzzle	GOOSE_ours	798.68	1.84E+07	110.4	1.66E+05	38%
	DCAF(LOO)	1013.88	8.62E+06	82.4	1E+04	50%
	DCAF	1414.23	2.06E+07	127.63	1.61E+05	**72%**
	FD(FF)	100.38	5.11E+05	7.61	3.22E+04	**98%**
	GOOSE	-	-	-	-	50%
sokoban	GOOSE_ours	104.59	1.8E+06	116.9	1.53E+04	63%
	DCAF(LOO)	103.4	2.54E+06	163.8	1.5E+04	65.5%
	DCAF	100.72	5.17E+06	258.4	2E+04	97%
	FD(FF)	-	-	-	-	-
	GOOSE	-	-	-	-	-
spanner	GOOSE_ours	-	-	-	-	-
	DCAF(LOO)	790.3	3.2E+06	210.0	1.52E+04	7%
	DCAF	765.9	1.32E+07	490.5	2.6E+04	**18%**
	FD(FF)	1842.14	2.77E+07	252.37	2.89E+04	8%
	GOOSE	-	-	-	-	18%
visitall	GOOSE_ours	1950.32	3.20E+07	340.02	9.41E+04	20%
	DCAF(LOO)	1800.91	4.5E+07	480	9.37E+04	27%
	DCAF	1850.07	5.3E+07	590.4	8.97E+04	**45%**
	FD(FF)	2110.69	1E+06	30	2.89E+04	29%
	GOOSE	-	-	-	-	81%
visitsome	GOOSE_ours	2100.45	9.00E+06	300.0	3.00E+04	64%
	DCAF(LOO)	2000.72	1.10E+07	330.0	3.33E+04	80%
	DCAF	1950.30	2.00E+07	550.0	3.64E+04	**85%**

Table 2: Comparison of DeepCubeAF (DCAF) with the domain-independent Fast Downward planner (FF heuristic), the GOOSE GNN-based heuristic learner, and the single-domain DCAF(LOO) is a leave-one-out experiment that does not use the given domain during training. Metrics include solution length, node expansions, search time (seconds), nodes expanded per second, and the percentage of solved problems. Note: All other metrics are computed based on solved instances only.

Domain	Solver	Len	Nodes	Secs	Nodes/Sec	Solved
folding	FD(FF)	512.1	8.40E+05	58.47	1.44E+04	38%
	DeepCubeA	420.5	9.00E+06	89.56	1.01E+05	**90%**
	GOOSE_ours	486.2	2.10E+06	120.20	1.75E+04	38%
	DCAF	600.8	4.80E+06	297.72	1.61E+04	60%

Table 3: Comparison of DeepCubeAF (DCAF) with the domain-independent Fast Downward planner (FF heuristic), the GOOSE GNN-based heuristic learner, and DeepCubeA, which is trained only on the folding domain. The folding domain is not seen during training for DCAF and GOOSE. Metrics include solution length, node expansions, search time (seconds), nodes expanded per second, and the percentage of solved problems. Note: All other metrics are computed based on solved instances only.

8 Conclusion

This paper introduces DeepCubeAF, the first foundation model based on deep reinforcement learning for learning a heuristic function that generalizes across pathfinding domains. DeepCubeAF is trained on randomly generated domains and problem instances and demonstrates improved generalization compared to heuristic functions trained with supervised learning and competitive performance with traditional domain-independent heuristic planners. Experimental results show that DeepCubeAF generalizes to both domains used during training and unseen domains.

References

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

Forest Agostinelli, Rojina Panta, and Vedant Khandelwal. Specifying goals to deep neural networks with answer set programming. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 2–10, 2024.

Oladayo S Ajani, Sung-ho Hur, and Rammohan Mallipeddi. Evaluating domain randomization in deep reinforcement learning locomotion tasks. *Mathematics*, 11(23):4744, 2023.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996. ISBN 1-886529-10-8.

Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: learning retrosynthetic planning with neural guided A* search. In *International Conference on Machine Learning*, pp. 1608–1616. PMLR, 2020.

Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20078–20086, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Vedant Khandelwal, Amit Sheth, and Forest Agostinelli. Pddlfuse: A tool for generating diverse planning domains. *arXiv preprint*, 2024.

Tianhua Li, Ruimin Chen, Borislav Mavrin, Nathan R Sturtevant, Doron Nadav, and Ariel Felner. Optimal search with neural networks: Challenges and approaches. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pp. 109–117, 2022.

Yuliang Li, Nitin Kamra, Ruta Desai, and Alon Halevy. Human-centered planning. *arXiv preprint arXiv:2311.04403*, 2023.

Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21(2):35–35, 2000.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pp. 1162–1176. PMLR, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J Joshi, et al. Robovqa: Multimodal long-horizon reasoning for robotics. *arXiv preprint arXiv:2311.00899*, 2023.

William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 574–584, 2020.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*, 2023.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Yuan-Hang Zhang, Pei-Lin Zheng, Yi Zhang, and Dong-Ling Deng. Topological quantum compiling with reinforcement learning. *Physical Review Letters*, 125(17):170501, 2020.