A Training Data Recipe to Accelerate A* Search with Large Language Models

Devaansh Gupta
devaansh@cs.ucla.edu
Nanyang Technological University
University of California, Los Angeles

Boyang Li
boyang.li@ntu.edu.sg
Nanyang Technological University

Abstract

Combining Large Language Models (LLMs) with heuristic search algorithms like A* holds the promise of enhanced LLM reasoning and scalable inference. To accelerate training and reduce computational demands, we investigate the coreset selection problem for the training data of LLM heuristic learning. Few methods to learn the heuristic functions consider the interaction between the search algorithm and the machine learning model. In this work, we empirically disentangle the requirements of A* search algorithm from the requirements of the LLM to generalise on this task. Surprisingly, we find an overlap between their requirements; A* requires more accurate predictions on search nodes near the goal, and LLMs need the same set of nodes for effective generalisation. With these insights, we derive a data-selection distribution for learning LLM-based heuristics. On three classical planning domains, maze navigation, Sokoban and sliding tile puzzles, our technique reduces the number of iterations required to find the solutions by up to $15\times$, with a wall-clock speed-up of search up to $5\times$. The codebase is at https://github.com/devaansh100/a_star.

1 Introduction

Contrary to the view of Large Language Models (LLMs) as a monolithic paradigm for intelligence, the dual-process theory of cognitive science (Stanovich and West, 2000; Kahneman, 2011) posits that human cognition consists of two closely collaborating systems, System 1 and System 2. System 1 exhibits typical traits of statistical learning such as fast inference and slow adaptation to novel problems. In comparison, System 2 can solve novel problems and excels at logical reasoning, but its inference speed is slow.

Recent analyses analogize LLMs to System 1 (Saha et al., 2024; Wang et al., 2024), as LLMs perform poorly at novel, out-of-distribution problem formulations (Wu et al., 2024) or problems that require planning and reasoning (Valmeekam et al., 2023; Tiong et al., 2024; Cheng et al., 2024; Kambhampati et al., 2024). On the other hand, tree-search methods like A* (Hart et al., 1968) and variants (e.g., Korf 1985; Kocsis and Szepesvári 2006), provide classic solutions to logical reasoning and planning, but they are unable to learn from past experiences and limited in speed due to sequential dependencies. Though it has been speculated that Artificial General Intelligence requires both System 1 and System 2 capabilities (Saha et al., 2024; Yu et al., 2024), how to fruitfully combining LLMs with search techniques remains an open problem.

We study the problem of using LLMs to learn A* heuristics, which are functions that estimate the distance from a search node to the goal state. However, it can be computational demanding to train LLMs and to generate training data, as ground-truth labels for training can only be obtained from successfully solved problems. With this paper, we aim to improve the efficiency of heuristic learning by selecting a small subset of training data, known as the coreset, which would lead to near-identical A* performance as the whole dataset. To the best of our knowledge, no previous work investigated the coreset selection problem for A* heuristic learning.

A complication of coreset selection in the A* + LLM setup is that the two algorithms may impose different requirements on training data. In this work, we attempt to disentangle and individually quantify the requirements of the two algorithms. We empirically test how different training data would change the generalisation of the LLM, and how A* reacts to generalisation errors in different positions of the search trajectory.

We divide the training trajectory into three equally sized portions: the beginning, the middle, and the end. First, we evaluate their effectiveness as LLM training data. This is inspired by research using training data difficulty as a metric for coreset

selection (Paul et al., 2021). A natural definition for difficulty in A* is the distance to goal, which indicates how many decisions must be made before reaching the goal. Intuitively, it should be more difficult to guess the exact distance to goal at a given search node if the search node is in fact farther away from the goal. Further, to simulate the effect of LLM noise on A*, we inject random errors into oracle heuristic values in the three portions and observe effects on the search length.

We obtain interesting and unexpected findings. For the LLM, training on the last portion, where the search node is closest to the goal and the distances are easiest to fit, leads to the best generalisation among the three portions. Unexpectedly, A* demonstrates a similar behavior; correct predictions on the end portion are the most beneficial to search efficiency, even though one might expect earlier decisions to be more important in pruning search nodes. These observations suggest that we should prioritise training data from the last portion, which would lead to overall good LLM generalisation and best accuracy on the end portion, which in turn accelerates search.

Accordingly, we devise a planner-aware sampling strategy for training data, which prioritises search nodes near the end. In addition, this sampling strategy is general enough to be combined with other coreset selection methods. The proposed strategy incurs, on average, 9.5% fewer A* search steps than uninformed baselines and, in some cases, outperforms models trained with double the amount of data.

Our contributions can be summarised as follows,

1. To the best of our knowledge, we are the first work to study the coreset selection problem for A* heuristic learning. Further, we propose a mathematical criterion to select training data based on their distance to goal.

2. We study the training data requirements for the generalisation of the the learned heuristic function and how heuristic errors affect A* performance, and identify a common requirement shared by the two algorithms.

3. Subsequently, we propose a general planner-aware technique to select training data for an LLM-based heuristic function. Our technique outperforms uniform pruning and existing baselines in extensive experiments.

2 Related Works

We review several research directions related to our work. For a tabular summary of the works, see Section A.5.

2.1 Learning Heuristics for Planning

Machine Learning Techniques Learning for planning problems that aims to reduce the search length can be traced back at least to Yoon et al. (2006); Fern et al. (2011). This task was posed as a regression problem, learned with neural networks (Arfaee et al., 2011; ús Virseda et al., 2013). Post their success, more recent works explored various neural architectures and objective functions for this problem (Chrestien et al., 2021; Groshev et al., 2018; Kirilenko et al., 2023). However, existing methods do not cater to specific requirements of the search algorithm.

Search-aware Techniques Some works consider the requirements of the search algorithm during learning; Yonetani et al. (2021); Vlastelica et al. (2019) reformulate each step of the planner as a differentiable function, which can be optimized with the loss calculated at the end of search. However, propagating gradient through time can be compute-intensive. Similarly, Speck et al. (2021); Orseau et al. (2023); Orseau and Lelis (2021) learn heuristics by performing reinforcement learning, which could require significant trial-and-error. In this work, we take an alternate data-centric approach to optimize training data. With this, we can lower the computational cost during training, while maintaining the quality of the learned heuristic.

2.2 Large Language Models in Search

Tree Creation by LLMs In contrast to our focus on LLMs as heuristic functions, previous works have also explored using LLMs as a world model that directly generates the action given the environmental state in search. Yao et al. (2024) uses such a framework to build a tree and traverses it with depth/breadth-first search, while Hao et al. (2023) extends it to Monte Carlo Tree Search (MCTS), where the LLM selects the tree node to be expanded and generates its children.

LLMs with External Planners Besides a heuristic, LLMs have been combined with external planners in various capacities. For instance, Valmeekam et al. (2023) uses an LLM with the

LPG planner (Gerevini et al., 2002), which iteratively corrects errors in a plan. Seeding LPG with an LLM plan has been shown to work better than a random plan. LLMs have also been used to translate tasks to formal languages for symbolic solvers like PDDL (Liu et al., 2023) and ASP (Yang et al., 2023). Combining such planners with LLMs has also been explored in dynamic settings to incorporate environment feedback (Guan et al., 2023; Dagan et al., 2023). While these works primarily use off-the-shelf LLMs to improve symbolic planners, our work aims to train an LLM.

Improving LLM-based Heuristics Shinn et al. (2024) improved LLM heuristics by incorporating failure states into the in-context-learning prompt. This has further been incorporated into tree-based frameworks (Zhou et al., 2023a). Such failure states are discovered during the course of solving a problem, and thus are restricted to that particular problem instance. In contrast, we aim to train a generic heuristic function that works for all problem instances in a domain. An alternate line of work (Lehnert et al., 2024; Gandhi et al., 2024) utilizes chain-of-thought prompting for LLM planning and trains the LLM on the traces of tree-search algorithms, implicitly learning an improved heuristic. In contrast, we explicitly learn the heuristic by supervised learning.

2.3 Optimising Training Data

Coreset Selection involves pruning the training dataset to only contain important datapoints, without a significant drop in performance. While various works exist for LLM pre-training (Paul et al., 2021; Marion et al., 2023; Abbas et al., 2023), to the best of our knowledge, we are the first work to study this in the context of heuristic learning. Our findings correlate with those of Zhou et al. (2023b); Sorscher et al. (2022); easier data is required for learning in the low-data regime.

3 Preliminaries

3.1 A* Search

A* is a tree-based search algorithm that aims to find a path between a start node and any goal node by building a tree $\mathcal{T}$. The algorithm is presented as Algorithm 1. The set of all tree nodes is denoted as $\mathcal{N}$. For each node n, A* search keeps track of two values, (i) historical cost $g(n)$, which is the distance between the start node and n and (ii) heuristic $h(n)$ which is an estimate of the true distance $h^*(n)$ between n and the closest goal node. Each node may be associated with a state $s(n)$. An action modifies the state, causing a transition to a new node. For the search, A* maintains two lists, the frontier list P_{frt} and the closed list P_{cld}. At the beginning, the closed list is empty and the frontier list is initialised with the start node. The search is terminated when either a goal state is encountered, or P_{frt} is empty. Each iteration performs two steps, described below.

Selection This step picks the most-promising leaf node in search tree, which has the least cost $f(n) = g(n)+h(n)$. All leaf nodes are stored in the frontier list. If the state of the selected node is equal to the goal state, the search is terminated. Else, the expansion step is performed.

Expansion This step adds new children nodes to the selected node, thereby expanding the search tree. A child node is only added to the search tree if and only if there does not exist a node with the same state in either the frontier, or the closed list, with a lower $f(\cdot)$ value. Finally, the selected node is moved from the frontier to the closed list.

We define the search length $\mathcal{S}$ of A* as the length of the closed list[1] after termination of the search. The use of $h(n)$ makes A* an *informed* search algorithm, significantly reducing the size of the closet list compared to uninformed search. The path from start to goal, defined as $\pi = (n_0, n_1 ... n_l)$, is the sequence of l nodes from the start node to the goal node. The start-to-goal path with minimum length is called the *optimal path*, denoted by π^*. A* guarantees that the resulting path will be optimal if the heuristic is *admissible*, i.e., $h(n) \leq h^*(n), \forall\, n \in \mathcal{N}$. It can be shown that with $h(\cdot) = h^*(\cdot)$ and non-trivial tie-breaking, A* will act as an optimal policy with $\mathcal{S} = |\pi^*|$. An inadmissible heuristic, however, does not necessarily create sub-optimal solutions.

3.2 Training Data for the Heuristic LLM

Our goal is train a language model θ, that, given a node n, can predict the residual $d^*(n) = h^*(n) - h(n)$ between the perfect heuristic $h^*(n)$ and a quick estimate $h(n)$. Given a series of similar problem instances, we derive training data from their A* search trees after a search is complete. For each tree node n, computing the ground-truth

[1] which is equal to the number of search iterations

Algorithm 1 A* Search

$P_{\text{frt}} \leftarrow \{n_{\text{start}}\}$
$P_{\text{cld}} \leftarrow \{\}$
while $|P_{\text{frt}}| > 0$ **do**
  $n \leftarrow \arg\min_{n \in P_{\text{frt}}} f(n)$ ▷ Selection
  if goal-state$(s(n))$ **then**
    return n
  end if
  for $c \in children(n)$ **do** ▷ Expansion
    $g(c) \leftarrow g(n) + 1$
    $f(c) \leftarrow g(c) + h(c)$
    if $(\nexists m \in P_{\text{frt}} \cup P_{\text{cld}}, s(c) = s(m))$ **or**
      $(\exists m \in P_{\text{frt}} \cup P_{\text{cld}}, s(c) = s(m)$ **and**
      $f(c) < f(m))$
    then
      Tree $\mathcal{T} \leftarrow \mathcal{T} \cup \{c\}$
      $P_{\text{frt}} \leftarrow P_{\text{frt}} \cup \{c\}$
    end if
  end for
  $P_{\text{frt}} \leftarrow P_{\text{frt}} - \{n\}$
  $P_{\text{cld}} \leftarrow P_{\text{cld}} \cup \{n\}$
end while

$d^*(n)$ would require running A* starting from node n, which quickly becomes prohibitively expensive as the problem size grows. Following Chrestien et al. (2021); ús Virseda et al. (2013), we only consider nodes on the optimal path. After the first A* run, their $h^*(\cdot)$ is trivial: for any node $n_j \in \pi^*$, $h^*(n_j) = |\pi^*| - j$. Formally, the training sequences $\mathcal{X}$ are given by $\mathcal{X} = \bigcup_{\pi_i^* \sim \Gamma_{i=0}^N} \{(n_j, d^*(n_j)), n_j \in \pi_i^*\}$.

3.3 Loss functions

We train the LLM with the L2 loss

$$\mathcal{L}_{L2} = (f_\theta(n) - d^*(n))^2 \tag{1}$$

where f_θ represents a forward pass of the LLM. We use encoder-decoder transformers and add a regression head ϕ_{L2} on the decoder that predicts $d^*(n)$ given the $\langle BoS \rangle$ token as the input.

Additionally, since the LLM can be trained in a text-to-text setting, we train a separate model with the canonical autoregressive loss, given by:

$$\mathcal{L}_{LM} = -\log p(d^*(n)|\theta) \tag{2}$$

With $\mathcal{L}_{LM}$, the pre-trained language model head ϕ_{LM} is used.

3.4 Inference

Inference involves leveraging the trained LLM in A* search. During the expansion step, children nodes to be evaluated are converted into an LLM prompt, from which the LLM predicts $d(n)$. This value is added to the quick estimate of $h(n)$. Notably, only a single forward pass is performed per expansion as we collate all children nodes as one batch. Additionally, we cache these prompts, such that if a state is revisited in another node m, $d(m)$ can simply be retrieved.

For θ trained with $\mathcal{L}_{LM}$, we perform top-k decoding, with $k = 5$[2], along with self-consistency (Wang et al., 2022), predicting 3 sequences, as this works slightly better in practice. The exact prompt inputs for the encoder have been provided in Section A.2.

3.5 Problem Domains

We conduct our experiments on three problems domains. Each domain comprises of the in-distribution (IID) and out-of-distribution (OOD) test sets for a total of six datasets.

Maze Navigation is a standard maze puzzle that involves finding an unobstructed path from the start to the goal state. The state of a node $s(\cdot)$ is characterized by the position of the player on the board. The quick admissible heuristic function used in the training data (and reference solutions) is the Manhattan distance between the player and the goal positions. Training and validation is performed on sequences derived from mazes of size 20×20. The IID test split consists of mazes of the same size, while OOD split consists of mazes of size 30×30.

Sokoban is a puzzle game involving a player pushing one or more boxes to fixed docks. This puzzle is considerably harder than maze, since a few wrong moves can lead to deadlocked states. The state of a node is characterized by the position of the player on the board, and the position of the boxes. Note that all boxes and docks are identical. The quick admissible heuristic function used is the sum of the minimum Manhattan distance between the player position and a box, and the sum of Manhattan distances between the boxes and their assigned docks. Boxes are assigned to docks by solving the minimum cost assignment

[2]This value was arbitrarily chosen and fixed for all experiments. It allows the LLM to make additional choices, without straying too much from the greedy one

problem with the Hungarian algorithm. Training, validation and IID testing is performed on 2-box problems, while OOD tests are on a mixture of problems with 2, 3 or 4 boxes.

Sliding Tile Puzzle (STP) is a puzzle consisting of a square board with distinct tiles and one empty space. The task is to move tiles into the empty space to reach a goal configuration. The state of a node is given by the current configuration of the board, and the quick admissible heuristic used is the sum of the Manhattan distance of each tile to its target position. Training, validation and the IID test sets comprise of 3×3 puzzles while the OOD test set consists of harder 4×4 and 5×5 puzzles.

The exact generation and composition of the datasets is described in Section A.1. In LLM prompts, we use ASCII encoding of the problems shown in Figures 2, 3 and 4.

3.6 Metrics

We adopt several metrics defined by Lehnert et al. (2024), (i) inverse-length-ratio (ILR) to measure the differences in the search length, (ii) success weighted by cost (SWC) to measure the differences in solution length and (iii) optimal %, to measure the percentage of problems solved optimally. ILR measures the average inverse ratio between the search length $\tilde{\mathcal{S}}$ of an A* solution, to the optimal reference $\mathcal{S}^*$. It is computed as

$$ILR = \frac{1}{N}\sum_{i=0}^{N}\frac{\mathcal{S}_i^*}{\tilde{\mathcal{S}}_i} \tag{3}$$

ILR can be averaged over various sets. ILR-on-solved is averaged over all puzzles in the test set and ILR-on-optimal is averaged over all puzzles whose solutions are optimal. Suboptimal solutions, found with inadmissible heuristics, are often discovered before optimal ones, leading to a lower $\mathcal{S}$, but a higher ILR; due to this, ILR-on-optimal allows us to measure the informativeness of the heuristic on equal, minimum length solutions.

SWC measures the average inverse ratio between the start-to-goal path length $|\tilde{\pi}|$ of an A* solution, to that of an optimal reference, denoted by $|\pi^*|$.

$$SWC = \frac{1}{N}\sum_{i=0}^{N}\frac{|\pi_i^*|}{|\tilde{\pi}_i|} \tag{4}$$

To measure computational cost, we propose a new metric, *inverse time ratio*, which is defined as the average inverse ratio between the wall-clock time of an A* solution $\tilde{WT}$ and a reference solution WT^*,

$$ITR = \frac{1}{N}\sum_{i=0}^{N}\frac{WT_i^*}{\tilde{WT}_i} \tag{5}$$

Set with $h^*(\cdot)$	σ	ILR-on-solved	ILR-on-optimal	SWC	Optimal %
All	-	2.7356	2.7356	1.0000	100
Initial	2	1.7314	1.7717	0.9896	84.9
Middle	2	1.8911	1.9309	0.9908	86.4
End	2	**2.2248**	**2.2617**	**0.9919**	**87.9**
Initial	4	1.0842	1.1912	**0.9530**	46.1
Middle	4	1.1604	1.2924	0.9516	46.2
End	4	**1.5439**	**1.7389**	0.9520	**46.3**
Initial	6	0.8579	0.9827	0.9229	28.6
Middle	6	0.9192	1.0811	**0.9232**	**29.3**
End	6	**1.2157**	**1.5287**	0.9202	28.1

Table 1: Experimental results with the oracle heuristic on the validation puzzles of maze navigation.

4 Disentangling A* and Heuristic Learning

4.1 Understanding Requirements of A*

Prediction errors by the LLM in the learned heuristic function are inevitable. In this section, we aim to examine two research questions: (i) how the prediction errors in the learned heuristic function affects the search length $\mathcal{S}$, and (ii) how they affect optimality of the solutions.

Specifically, we start with the oracle heuristic $h^*(n)$ and artificially introduce error in different sections of the search trajectory in order to observe effects on $\mathcal{S}$ and optimality. The search tree is divided into three sets—*initial*, *middle* and *end*. A node n is placed in the initial set if its cost places itself in the first third of the optimal path: $g(n) < |\pi^*|/3$. Alternatively, it may be placed in the middle set if $|\pi^*|/3 \leq g(n) < 2|\pi^*|/3$, and in the end set if $g(n) \geq 2|\pi^*|/3$. We introduce zero-mean Gaussian error by drawing a random value from $\mathcal{N}(0, \sigma)$ and adding it to $h^*(n)$. In each experiment, we introduce errors in two of three sections and use the oracle in one section. We use maze as the domain of experiment and obtain the oracle heuristic $h^*(\cdot)$ by running Dijkstra's algorithm on the maze, starting from the goal.

Results The results are shown in Table 1. The rows *All*, *Initial*, *Middle*, and *End* indicate the tree section where the oracle is utilized, and *All* means the oracle is always used. Clearly, the oracle heuristic gives the best performance, but that is not easy

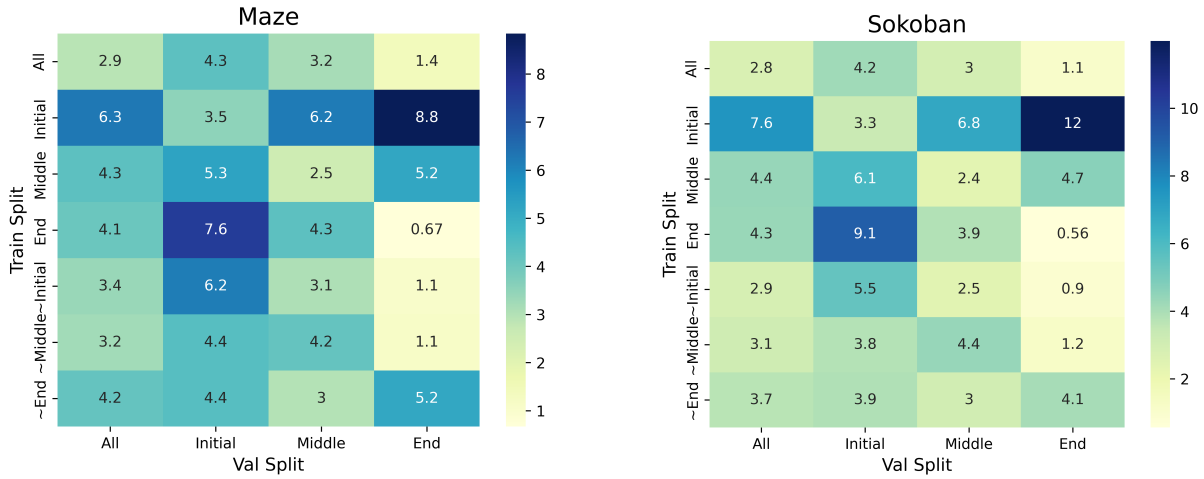

Figure 1: Validation MAE of models trained on the *Initial*, *Middle*, *End*, and *All* splits, and their corresponding exclusion sets. A lower value shows better generalisation.

Test Splits →		**IID**				**OOD**			
Train Split	**Domain**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**
All	Maze	1.5666	1.5654	0.9972	97.60	1.3320	1.3309	0.9965	96.00
Initial		0.9101	0.9101	**1.0000**	**100.0**	0.8193	0.8193	**1.0000**	**100.0**
Middle		0.8370	0.837	**1.0000**	**100.0**	0.8059	0.8059	**1.0000**	**100.0**
End		**1.2081**	**1.2033**	0.9974	97.40	**1.1018**	**1.1055**	0.9957	95.40
∼ Initial		1.2117	1.2132	0.9989	99.00	1.0581	1.0594	0.9992	98.80
∼ Middle		**1.6053**	**1.6151**	0.9907	92.80	**1.2476**	**1.2360**	0.9950	94.40
∼ End		0.9202	0.9202	**1.0000**	**100.0**	0.9198	0.9198	**1.0000**	**100.0**
All	Sokoban	8.3800	8.8785	0.9761	73.94	11.1967	11.7906	0.9815	74.46
Initial		0.6658	0.6661	**0.9967**	**93.66**	0.5940	0.5917	0.9956	90.12
Middle		0.9710	1.0049	0.9901	83.80	0.8148	0.8399	0.9904	84.34
End		**3.0312**	**3.0642**	0.9965	**93.66**	**2.7465**	**2.7721**	**0.9986**	**96.39**
∼ Initial		6.1912	6.5422	**0.9862**	**82.04**	9.2832	9.8333	**0.9893**	**83.86**
∼ Middle		**9.7389**	**9.9559**	0.9578	56.69	**16.3567**	**18.1764**	0.9650	61.45
∼ End		2.8397	2.9638	0.9854	80.28	2.9484	3.0910	0.9854	78.07

Table 2: Results from LLM heuristics trained on different data splits, demonstrating the importance of the *End* split for generalisation to A* search on both maze and Sokoban.

to achieve by a learned model. Amongst other experiment conditions, with the same σ, using $h^*(\cdot)$ on nodes in the end set performs the best on both ILR-on-solved and ILR-on-optimal. Moreover, the absolute differences in performance by using $h^*(\cdot)$ in the middle and end sets are larger than the differences between middle and initial. These performance gaps are larger with a higher σ.

There does not seem to be a clear trend between SWC and Optimal % amongst the three sets. Both metrics go down with increasing σ. This is not surprising, since with higher error, the heuristic will be inadmissible more frequently, thereby increasing the probability of finding longer, suboptimal solutions.

Implications The most important implication of these experiments is that, if we can only minimize errors of the heuristic function on one section of the search trajectory, we should choose the end section, which is closest to the goal. Doing so yields the highest ILR. Speculatively, erroneous decisions earlier in the trajectory may be corrected later, if we can make good decisions near the end of the search process.

4.2 Understanding Generalisation of Heuristic Learning

In this section, we explore how training on pairs of (node, distance-to-goal) affects the generalisation of the heuristic-learning LLM. We create four training splits by uniformly sampling nodes on the optimal path from the *Initial*, *Middle*, and *End* sections of the path. The *All* set contains nodes uniformly sampled from all three sections. Addi-

tionally, we also create exclusion sets, which excludes one of the three sections, and these sets are denoted as $\sim$*Initial*, $\sim$*Middle* and $\sim$*End*. For instance, $\sim$*Initial* contains only data sampled from the *Middle* and *End* sets. All training splits have the same size.

We adopt the following evaluation metrics: (i) mean absolute error (MAE) on validation splits containing nodes from each of the aforementioned splits, and (ii) ILR achieved by applying the trained models as heuristic functions for A*. While (i) directly evaluates the generalisation of the model, (ii) provides a more realistic test of how well the trained model works with A*.

Each training split contains 12k and 8k nodes in total for maze and Sokoban, respectively. All models are initialized with code-t5-small. Hyperparameter details are mentioned in Section A.3.

Results The LLM generalisation results are shown in Figure 1 and results from applying different LLMs with A* are shown in Table 2. First, as we expect, each split generalises the best to itself, but shows poor generalisation to the others. *All* achieves the best generalisation to each split. Second, on ILR, *End* performs the best when combined with A*. However, this is still inferior to the performance of *All*. This is consistent with the trends observed in Section 4.1.

Amongst the exclusion sets, we observe that $\sim$*End* achieves the worst generalisation and the worst ILR in both domains and both IID and OOD test splits. The comparison between the other two sets is mixed. $\sim$*Middle* has the best ILR performance, whereas $\sim$*Initial* performs well on Optimal % and SWC.

Implications Heuristics learned from the end set performs the best on MAE and well on ILR, showing that we need the end set in the training mix. These nodes can be considered easier than others because it is easy to foresee the distance to goal for a node positioned near the goal. However, the good performance of $\sim$*Middle* and $\sim$*Initial* suggests that easy nodes by themselves are not enough, and we should expose the model to some difficult nodes from the other sets, which are further away from the goal.

5 Proposed Solution

The Utility of a Node in Accelerating Search Inspired by experiments in Section , we propose to quantify the utility of a node in reducing the search length as,

$$\mathcal{C}(n) = \log\left(\frac{|\pi^*|}{|\pi^*| - g(n)}\right) \tag{6}$$

$\mathcal{C}(\cdot)$ assigns higher values to nodes closer to the goal. While there can be nodes with $g(n) \geq |\pi^*|$, since they are never added to the tree, $\mathcal{C}(\cdot)$ is not defined for them. Considerations and other choices for $\mathcal{C}(\cdot)$ are discussed in Section A.4.

Planner-aware Sampling We have shown that accurate prediction of the heuristic for nodes near the goal will lead to maximal reduction of the search length. Additionally, we want to include nodes from the initial and middle sets as well, to optimize ILR performance. Thus, we propose to sample from a distribution $\mathcal{D}(\cdot)$ that prioritises these nodes, based on Equation 6 (as opposed to a uniform distribution), given by,

$$\mathcal{D}(n, \tau) = SoftMax\left(\frac{1}{\tau}\mathcal{C}(n)\right), \forall n \in \pi^* \tag{7}$$

where τ denotes temperature. Increasing τ increases the hardness of the training dataset, thereby increasing the number of nodes sampled from the initial and middle sets.

Combining with Baselines Planner-aware sampling can be easily combined with any coreset selection baseline to enhance it for this task. This is done by first sampling two sets of nodes (without replacement), once using any coreset selection baseline Ψ, and another with $\mathcal{D}(n, \tau)$. Post this, the nodes can be resampled, without replacement, from the union of these two sets, where nodes appearing in both the sets are twice as likely to get sampled than those appearing in only a single set. This procedure is summarised in Algorithm 2.

Algorithm 2 Combining planner-aware sampling with a coreset selection baseline Ψ.

Assume m nodes are sampled from a problem,

$\mathcal{S}_1 \leftarrow \{n_i \sim \Psi(n) \mid i \in [1, m]\}$

$\mathcal{S}_2 \leftarrow \{n_i \sim \mathcal{D}(n, \tau) \mid i \in [1, m]\}$

$$\mathcal{P}(n_i) \leftarrow \begin{cases} \frac{2}{|\mathcal{S}_1|+|\mathcal{S}_2|}, & n_i \in \mathcal{S}_1 \cap \mathcal{S}_2 \\ \frac{1}{|\mathcal{S}_1|+|\mathcal{S}_2|}, & otherwise \end{cases}$$

$\mathcal{X} \leftarrow \{n_i \sim \mathcal{P}(\mathcal{S}_1 \cup \mathcal{S}_2)) \mid i \in [1, m]\}$

6 Experiments

Baselines The proposed sampling method is denoted as $\mathcal{D}(n, \tau)$. The *full-data* baseline trains on

Test Splits →		IID				OOD			
Train Split	**Domain**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**
Full-data	Maze	1.6739	1.6756	0.9967	97.0	1.2755	1.2730	0.9967	95.8
$\mathcal{X} \sim \mathcal{U}(n)$		1.5666	1.5654	**0.9972**	**97.6**	1.3320	1.3309	0.9965	96.0
$\mathcal{X} \sim \mathcal{D}(n, 2)$		1.7029	1.7035	0.9958	96.6	1.3365	1.3354	0.9964	95.0
$\mathcal{X} \sim SD$		1.6412	1.6453	0.9941	95.2	1.2823	1.2821	**0.9980**	**97.6**
$\mathcal{X} \sim SD + \mathcal{D}(n, 2)$		**1.7182**	**1.7245**	0.9927	94.6	**1.3568**	**1.3521**	0.9968	96.4
Full-data	Sokoban	11.6416	12.5933	0.9834	79.93	14.7093	15.2655	0.9847	77.83
$\mathcal{X} \sim \mathcal{U}(n)$		8.3800	8.8785	0.9761	73.94	11.1967	11.7906	0.9815	**74.46**
$\mathcal{X} \sim \mathcal{D}(n, 0.8)$		10.2077	10.8168	**0.9808**	**75.70**	13.7706	13.7546	**0.9828**	77.11
$\mathcal{X} \sim SD$		10.8579	11.5282	0.9702	68.66	14.9133	15.4475	0.9757	71.58
$\mathcal{X} \sim SD + \mathcal{D}(n, 5)$		**11.5184**	**11.8487**	0.9732	68.66	**15.8553**	**15.9748**	0.9772	72.05
Full-data	STP	4.1509	4.5750	0.9806	77.4	1.5012	1.5374	0.9860	84.4
$\mathcal{X} \sim \mathcal{U}(n)$		3.4040	3.7777	0.9755	72.8	1.3054	1.3789	0.9859	85.2
$\mathcal{X} \sim \mathcal{D}(n, 5)$		3.4758	3.9686	**0.9765**	**73.8**	1.4265	1.4606	**0.9946**	**93.0**
$\mathcal{X} \sim SD$		3.5372	4.2400	0.9617	60.6	**2.4353**	**2.7080**	0.9804	77.4
$\mathcal{X} \sim SD + \mathcal{D}(n, 5)$		**4.2779**	**4.7384**	0.9723	70.6	1.7050	1.8955	0.9694	69.6

Table 3: Experimental results with $\mathcal{L}_{L2}$ by sampling from the $\mathcal{D}(n, \tau)$ distribution. Best scores are in **bold**.

Test Splits →			IID				OOD			
Base Model	**Train Split**	**Domain**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**
codet5-base	$\mathcal{X} \sim \mathcal{U}(n)$	Maze	1.7218	1.7245	**0.9965**	**97.0**	1.2841	1.2722	0.9970	96.4
	$\mathcal{X} \sim \mathcal{D}(n, 2)$		**1.8112**	**1.8142**	0.9957	96.8	**1.3460**	**1.3422**	**0.9977**	**97.2**
codet5-large	$\mathcal{X} \sim \mathcal{U}(n)$		1.2963	1.2966	**0.9995**	**99.6**	1.1531	1.153	**0.9994**	**99.2**
	$\mathcal{X} \sim \mathcal{D}(n, 1)$		**1.6920**	**1.6982**	0.9964	97.4	**1.3101**	**1.3088**	0.9980	97.6
t5-small	$\mathcal{X} \sim \mathcal{U}(n)$		1.5447	1.5483	**0.9967**	**97.2**	1.3287	1.3276	**0.9975**	**97.0**
	$\mathcal{X} \sim \mathcal{D}(n, 2)$		**1.5785**	**1.5818**	0.9957	96.4	**1.3404**	**1.3378**	0.9974	**97.0**
codet5-base	$\mathcal{X} \sim \mathcal{U}(n)$	Sokoban	**10.8858**	11.1579	0.9770	71.83	14.4553	14.4831	0.9810	74.70
	$\mathcal{X} \sim \mathcal{D}(n, 2)$		10.6828	**11.1692**	**0.9791**	**73.94**	**15.0611**	**15.2904**	**0.9828**	**76.39**
codet5-large	$\mathcal{X} \sim \mathcal{U}(n)$		10.3732	**10.7997**	0.9788	74.3	12.8759	12.9480	0.9830	76.39
	$\mathcal{X} \sim \mathcal{D}(n, 2)$		**10.3778**	10.7343	**0.9850**	**80.99**	**13.0179**	**12.9534**	**0.9891**	**83.37**
t5-small	$\mathcal{X} \sim \mathcal{U}(n)$		10.8294	**11.1671**	0.9707	70.07	11.4536	11.2696	**0.9882**	**80.96**
	$\mathcal{X} \sim \mathcal{D}(n, 2)$		**10.9260**	10.9835	**0.9803**	**75.00**	**12.4921**	**12.7784**	0.9865	78.80

Table 4: Experiments with $\mathcal{L}_{L2}$, showing the effects of planner-aware sampling on various models.

all nodes (22.3k nodes for maze, 26.3k for Sokoban and 23.7k for STP) on the optimal path without subsampling. The uniform sampling baseline $\mathcal{U}(n)$ gives equal probability to all nodes. To the best of our knowledge, there are no search-aware coreset selection methods. Hence, we adopt as a baseline an LLM-based coreset selection method, SemDeDup (SD) (Abbas et al., 2023), which discards semantically similar data points from the training dataset. On top of SD, we apply Algorithm 2 to make it search-aware ($SD + \mathcal{D}(n, \tau)$). All coreset selection methods select 8k nodes for STP, and as before, 8k for Sokoban and 12k for maze.

Results The results using the $\mathcal{L}_{L2}$ loss are shown in Table 3. We defer results with $\mathcal{L}_{LM}$ to Table 10. $\mathcal{D}(n, \tau)$ consistently outperforms uniform sampling on ILR by an average of 4.4% on maze, 5.7% on STP, and a much larger margin of 12.5% on Sokoban. On maze, $\mathcal{D}(n, \tau)$ also outperforms the full-data baseline on OOD data, which is trained on 46.5% more data points. These results also extend to $\mathcal{L}_{LM}$, where $\mathcal{D}(n, \tau)$ outperforms $\mathcal{U}(n)$ by an average of 5%.

In terms of metrics of solution optimality (SWC and Optimal %), $\mathcal{D}(n, \tau)$ remains competitive and is better than the baselines by an average of 0.24%. Interestingly, training on all the data gives higher performance on optimality metrics, which could be a consequence of lower validation error, due to more training data.

Notably, the SD coreset selection baseline, developed for LLMs, also performs quite well. However, SD augmented with $\mathcal{D}(n, \tau)$ outperforms all other methods, except on STP OOD, by an average of 8.75%.

Model Scale and Pre-training To test the effectiveness of our method while scaling up the LLM, we experiment with thre LLMs, t5-small (60M), codet5-base (220M), and codet5-large (770M). Table 4 demonstrates similar trends of $\mathcal{D}(n, \tau)$ outperforming $\mathcal{U}(n)$. Interestingly, the performance of larger models is not always better than that of smaller models. This could be attributed to the fact that our experiments have been performed in the low-data regime and large models cause overfitting. Studying the effects of scaling up data with param-

eters is left for future works. The learned heuristics with larger models are more optimal, suggesting less error in the predictions.

Time Cost of LLM Inference It is well accepted in the planning domain that a more informative heuristic is more expensive to compute (Bylander, 1994). While LLMs incur additional time during inference, the learned heuristic is informative enough to amortize the extra time cost. We use ITR as the evaluation metric, which shows speed-ups in wall-clock search time compared to the LLM-free A* search. An ITR value > 1 implies that the LLM heuristic is faster than the base heuristic.

Experiments are performed on the $\mathcal{D}(n, \tau)$ models (from Table 3), trained with $\mathcal{D}(n, \tau)$ sampling. We show the results in Table 5. Due to its difficulty, Sokoban has a high number of explored nodes in each problem (often greater than 10k). With the LLM heuristic, the ITR on the most difficult OOD test split is greater than one. On the IID set, with easier problems of shorter search lengths, the ITR is close to one, but does not surpass it. Similarly, the ITR is less than one on maze, which consists of easier problems with low $\mathcal{S}$. Since the number of nodes is already quite low (mostly between 2k and 2.5k), a reduction does not necessarily bring about wall-clock speed-up. With this, we conclude that the LLM search heuristic is the most beneficial on hard OOD problems, which is also where direct inference from LLMs struggle the most (Wu et al., 2024) and exactly what is needed for boostrapping from easy to hard problems.

Interestingly, $\mathcal{L}_{LM}$ is almost $2.5\times$ slower on average than $\mathcal{L}_{L2}$, despite the ILR being only $1.1\times$ worse. This suggests that though ϕ_{LM} is capable of learning an informative heuristic, the forward pass through the larger linear layer, along with stochastic decoding, negatively affects efficiency.

Training Target Between $\mathcal{L}_{L2}$ and $\mathcal{L}_{LM}$ models, the former consistently outperforms the latter on the IID test split, while on OOD, the results are mixed, with $\mathcal{L}_{LM}$ being slightly better, at least with uniform sampling on Sokoban. $\mathcal{L}_{LM}$ is more aligned with the pre-training of the base model, which may have improved generalisation beyond the training data distribution.

Another interesting observation is that the best hyperparameter τ used with $\mathcal{D}(n, \tau)$, tuned on the validation set, is usually higher for $\mathcal{L}_{LM}$, suggesting that it has a higher preference for data points in the initial set, which can be considered harder than other nodes.

Domain	Test Split	ITR-on-solved	ITR-on-optimal
Model : $\mathcal{L}_{L2}$			
Sokoban	IID	0.8167	0.8735
	OOD	5.9441	5.9215
Maze	IID	$5.122e-3$	$5.127e-3$
	OOD	$5.062e-3$	$5.079e-3$
Model : $\mathcal{L}_{LM}$			
Sokoban	IID	0.2626	0.2611
	OOD	2.7250	2.3978
Maze	IID	$1.958e-3$	$1.963e-3$
	OOD	$2.090e-3$	$2.096e-3$

Table 5: Speed-ups in wall-clock search time achieved by using the trained language model as a heuristic.

7 Conclusion

In this work, we study the training data requirements to learn a strong heuristic for A* search. We find that accurate prediction of heuristics for nodes close to the goal are the most important for A* speed. Similarly, generalization of the LLM heuristic requires training on nodes near the goal. Based on these insights, we propose a mathematical formula to select search nodes as training data. This results in substantially reduced search lengths and significant wall-clock speedups on hard problems. Our study lays the groundwork for bootstrapped heuristic learning, which learns heuristic functions for increasingly larger problems using solved problems of smaller sizes. Referred to as the data flywheel, such techniques hold promise to scale up the capabilities of LLM + tree search[3].

Acknowledgments

We gratefully acknowledge the support by the Nanyang Associate Professorship, the National Research Foundation Fellowship (NRF-NRFF13-2021-0006), Singapore, and the Alibaba-NTU Global e-Sustainability CorpLab (ANGEL) under Project I2301E0026. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the funding agencies.

Limitations

Our study is restricted to classical puzzle domains, maze, Sokoban and STP. While we expect our

[3] https://twitter.com/DrJimFan/status/1834279865933332752

domain-independent analysis to generalise to other problems, experimental verification would be necessary to verify that conjecture. Moreover, since our work focuses on language models used as heuristics, it inherits the bias and fairness concerns associated with language models, which should be taken into consideration when deploying such models.

References

Amro Kamal Mohamed Abbas, Kushal Tirumala, Daniel Simig, Surya Ganguli, and Ari S Morcos. 2023. Semdedup: Data-efficient learning at web-scale through semantic deduplication. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Shahab Jabbari Arfaee, Sandra Zilles, and Robert C Holte. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17):2075–2098.

Tom Bylander. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204.

Ziru Chen, Michael White, Raymond Mooney, Ali Payani, Yu Su, and Huan Sun. 2024. When is tree search useful for llm planning? it depends on the discriminator. *arXiv preprint arXiv:2402.10890*.

Kewei Cheng, Jingfeng Yang, Haoming Jiang, Zhengyang Wang, Binxuan Huang, Ruirui Li, Shiyang Li, Zheng Li, Yifan Gao, Xian Li, Bing Yin, and Yizhou Sun. 2024. Inductive or deductive? rethinking the fundamental reasoning abilities of llms. *arXiv Preprint 2408.00114*.

Leah Chrestien, Tomas Pevny, Antonin Komenda, and Stefan Edelkamp. 2021. Heuristic search planning with deep neural networks using imitation, attention and curriculum learning. *arXiv preprint arXiv:2112.01918*.

Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*.

Marco Ernandes, Marco Gori, et al. 2004. Likely-admissible and sub-symbolic heuristics. In *ECAI*, volume 16, page 613. Citeseer.

Alan Fern, Roni Khardon, and Prasad Tadepalli. 2011. The first learning track of the international planning competition. *Machine Learning*, 84:81–107.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. 2024. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*.

Alfonso Gerevini, Ivan Serina, et al. 2002. Lpg: A planner based on local search for planning graphs with action costs. In *Aips*, volume 2, pages 281–290.

Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. 2018. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 408–416.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. 2019. An investigation of model-free planning. In *International conference on machine learning*, pages 2464–2473. PMLR.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

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

Daniel Kahneman. 2011. *Thinking, fast and slow.* Farrar, Straus and Giroux.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks.

Daniil Kirilenko, Anton Andreychuk, Aleksandr Panov, and Konstantin Yakovlev. 2023. Transpath: Learning heuristics for grid-based pathfinding via transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12436–12443.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

R. E. Korf. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97– 109.

Lucas Lehnert, Sainbayar Sukhbaatar, Paul Mcvay, Michael Rabbat, and Yuandong Tian. 2024. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Max Marion, Ahmet Üstün, Luiza Pozzobon, Alex Wang, Marzieh Fadaee, and Sara Hooker. 2023. When less is more: Investigating data pruning for pretraining llms at scale. *arXiv preprint arXiv:2309.04564*.

Laurent Orseau, Marcus Hutter, and Levi HS Lelis. 2023. Levin tree search with context models. *arXiv preprint arXiv:2305.16945*.

Laurent Orseau and Levi HS Lelis. 2021. Policy-guided heuristic search with guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12382–12390.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607.

Swarnadeep Saha, Archiki Prasad, Justin Chih-Yao Chen, Peter Hase, Elias Stengel-Eskin, and Mohit Bansal. 2024. System-1.x: Learning to balance fast and slow planning with language models. *arXiv Preprint 2407.14414*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536.

David Speck, André Biedenkapp, Frank Hutter, Robert Mattmüller, and Marius Lindauer. 2021. Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 597–605.

Keith E. Stanovich and Richard F. West. 2000. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and Brain Sciences*, 23(5):645–665.

Takeshi Takahashi, He Sun, Dong Tian, and Yebin Wang. 2019. Learning heuristic functions for mobile robot path planning using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 764–772.

Anthony Tiong, Junqi Zhao, Boyang Li, Junnan Li, Steven Hoi, and Caiming Xiong. 2024. What are we measuring when we evaluate large vision-language models? an analysis of latent factors and biases. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3427–3454, Mexico City, Mexico. Association for Computational Linguistics.

Jes ús Virseda, Daniel Borrajo, and Vidal Alcázar. 2013. Learning heuristic functions for cost-based planning. *Planning and Learning*, 4.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. 2019. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*.

Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. 2024. Litesearch: Efficacious tree search for llm.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *NAACL*.

Zhun Yang, Adam Ishay, and Joohyung Lee. 2023. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. 2021. Path planning using neural a* search. In *International conference on machine learning*, pages 12029–12039. PMLR.

Sung Wook Yoon, Alan Fern, and Robert Givan. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*, volume 2, page 3.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. 2024. Distilling system 2 into system 1. *arXiv 2407.06023*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023a. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. 2023b. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*.

A Appendix

A.1 Data Generation

Maze We generate mazes with a modified Prim's algorithm[4]. The start and goal states are randomly chosen until the following criteria are met, (i) length of the optimal plan > O_l, (ii) ratio between length of closed set after search and length of optimal plan is > $\alpha = 3.5$. If either of these are not met within 10 tries, a new maze is generated. Criterion (i) ensures that the start and goal positions are not too close and (ii) ensures that there are sufficient number of additional expanded nodes. It serves as a surrogate for the measure of hardness $h*(n_s)/h(n_s)$ where n_s is the start node, proposed in Takahashi et al. (2019). The surrogate is used since it is more aligned with the chosen metrics (ILR) in this work. However, this method only creates a maze with a single path to the goal. To get multiple paths, each node is designated to either be closer to the start, or to the goal, and walls are randomly broken at the boundary of these groups[5].

Sokoban This dataset is adapted from the open-source boxoban dataset, proposed in Guez et al. (2019). For the training puzzles, we randomly shuffle the provided training set from the "unfiltered" split, followed by subsampling B boxes per puzzle. We use the same filters as maze, but with different hyperparameters. The IID test split uses the same criteria, but samples puzzles from the testing set of boxoban. To reduce the data creation time, we constrain the number of iterations required by A* to solve a puzzle between β_{min} and β_{max}. The OOD split is curated to contain a mix of harder puzzles with varying number of boxes, length of optimal plans and higher number of iterations. All puzzles have size 10×10, $O_l = 20$ and $\alpha = 6$.

STP We generate 3×3 puzzles by randomly generating a sequence of tiles, checking if it is solvable with A*. For puzzles with a width greater than 3, we start from the goal configuration and perform 20 - 30 random moves to scramble the puzzle, from the goal state. For all puzzles, $\alpha = 6$, $O_l = 20$, $\beta_{min} = 0$ and $\beta_{max} = 5k$. To keep the symbols in the puzzle uniform between training and inference, the generation of puzzles is done with digits, however, they are fed to the model as alphabets. For each puzzle, we uniformly sample without replacement the required number of alphabets, sort them alphabetically and assign them to the digits.

The exact statistics are in Table 6, Table 7 and Table 8.

Split	# puzzles	Size	O_l
Train	750	20×20	20
Val	750	20×20	20
Test IID	500	20×20	20
Test OOD	500	30×30	30

Table 6: Dataset statistics for maze.

Split	# puzzles	B	β_{max}	β_{min}
Train	1000	2	7k	0
Val	1000	2	7k	0
Test IID	284	2	7k	0
Test OOD	15	2	14k	7k
	100	3	7k	0
	100	3	14k	7k
	100	4	7k	0
	100	4	14k	7k

Table 7: Dataset statistics for Sokoban.

Split	# puzzles	Size
Train	1000	3×3
Val	1000	3×3
Test IID	500	3×3
Test OOD	250	4×4
	250	5×5

Table 8: Dataset statistics for STP.

A.2 Prompts

The language models have been trained on a regression task with context prompts, which are provided below. Since the experiments are performed with code models, we tailor the prompt accordingly. The same prompt is used for both domains, shown in

[4] https://github.com/john-science/mazelib

[5] https://stackoverflow.com/a/22308159

```
##########
#    ######
#    # ##.#
# .   $    #
#       $ #
#   #######
#@ #######
#   #######
#   #######
##########
legend =  "@ - player, # - wall,
   . - empty docks, ' ' - empty
   cell, $ - box, X - box on dock
   , O - player on dock"
```

Figure 2: Puzzle representation and legend of a training puzzle from Sokoban.

Figure 5, with the puzzle representations and legend in Figure 2 for Sokoban and Figure 3.

A.3 Hyperparameters and Model Choice

All models are trained for 40 epochs, with a learning rate of $1e-4$, batch size of 64 and optimized with Adafactor. We implement early stopping, with the model chosen by best performance on validation MAE, computed every epoch. Training is performed on 1 NVIDIA A6000 Ada GPU.

Codet5-small was chosen for experiments since, (i) it is a compute-efficient, powerful LM, and (ii) we believed the code-pretraining would be beneficial to the code-like representation of our problem.

A.4 Additional Ablations

Choice of $\mathcal{C}(\cdot)$ Theoretically, any monotonically increasing function can be used for $\mathcal{C}(\cdot)$. Practically, however, some factors need to taken care of. For instance, we cannot use $e^{g(n)}$, since it's large first derivative will assign a very high contribution value to nodes near the goal. Thus, when used for sampling, it will concentrate all the probability mass near the goal, preventing us from augmenting the training set with harder nodes, further away from the goal.

We show additional results for two more choices for $\mathcal{C}(n)$, used in $\mathcal{D}(n, \tau)$, in Table 9. Note that the same τ used in the main body has been chosen, and is not tuned. Despite that, we outperform uniform sampling on most splits. This validates the general idea of using an increasing function for $\mathcal{C}(n)$. Choosing the best performing or most theoretically justified one is left for future works.

```
####################
#..@................#
###.#####.###.#######
#...#...#.#.#...#...#
#######.#.#.#.#####.#
#..........#.......#
###.#.#.#.#.#.#.#.#.#
#...#.#.#.#.#...#...#
#.#.#.#####.#.#.#.#.#
#.#...#.#..........#
###.#.#.#.#.###.#.#.#
#.........#...#.....#
#.#.#.#.#.#####.#.#.#
#.#.#.#.#.#.....#.#.#
#.###.#######.#.#.#.#
#...#.#X......#.....#
#.###.#.#.#.#.#.#.#.#
#...#.#.#.#.#.#.#...#
###.#####.###.#.###.#
#...#.....#...#.....#
####################
legend =  "@ - player, # - wall,
   . - empty cell, X - goal"
```

Figure 3: Puzzle representation and legend of a training puzzle from the maze dataset.

```
puzzle_str = "i a h m v o u 0 y"
goal = "0 a h i m o u v y"
legend =  "0 - empty space"
```

Figure 4: Puzzle representation and legend of a training puzzle from the stp dataset.

A.5 Summary of Related Works

A summary of the related works has been provided in Table 11.

```
import torch
def get_improved_heuristic(heuristic: int, difference: int):
    '''
        A function that takes in the admissible A* heuristic and adds
    to it the difference, to return a heuristic closer to the optimal
    cost to the goal. The difference should be calculated keeping in
   mind the optimal cost of the puzzle.
    '''
    return heuristic + difference

# The difference is calculated by observing the {domain} puzzle and
   deducing the optimal cost to goal. The heuristic is subtracted
   from this optimal cost
# {puzzle_legend}
puzzle_str = "{puzzle_str}"
improved_heuristic = get_improved_heuristic({heuristic},
```

Figure 5: Prompt used while training the language model. {curly braces} denote a placeholder.

Test Splits $\rightarrow$		**IID**				**OOD**			
$\mathcal{C}(n)$	**Domain**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**
$\log(\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)})$	Sokoban	10.2077	10.8168	0.9808	75.70	13.7706	13.7546	0.9828	77.11
$\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)}$		7.7467	7.7455	0.9806	78.87	11.9533	12.3032	0.9874	82.17
$\frac{g(n)}{\lvert\pi\rvert^*}$		9.2398	9.9242	0.9787	74.65	11.5371	11.9224	0.9846	80.24
$\log(\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)})$	Maze	1.7029	1.7035	0.9958	96.6	1.3365	1.3354	0.9964	95.0
$\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)}$		1.6119	1.6129	0.9961	96.6	1.2972	1.2949	0.9982	97.8
$\frac{g(n)}{\lvert\pi\rvert^*}$		1.6560	1.6553	0.9964	96.8	1.2691	1.2706	0.9968	96.2
$\log(\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)})$	STP	3.4758	3.9686	0.9765	73.8	1.4265	1.4606	0.9946	93.0
$\frac{\lvert\pi\rvert^*}{\lvert\pi\rvert^* - g(n)}$		3.0416	3.4088	0.9758	72.4	1.7935	1.8943	0.9885	86.6
$\frac{g(n)}{\lvert\pi\rvert^*}$		3.6157	4.0441	3.7528	95.4	1.4051	1.4421	0.9865	87.0

Table 9: Experimental results by sampling from the $\mathcal{D}(n, \tau)$, with different choices for $\mathcal{C}(\cdot)$, with the $\mathcal{L}_{L2}$ model.

Test Splits $\rightarrow$		**IID**				**OOD**			
Train Split	**Domain**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**	**ILR-on-solved**	**ILR-on-optimal**	**SWC**	**Optimal %**
Full-data	Maze	1.4752	1.4902	0.9925	94.0	1.2448	1.2467	0.9965	96.2
$\mathcal{X} \sim \mathcal{U}(n)$		1.4979	1.5070	**0.9897**	**92.2**	1.1869	1.1769	0.9925	92.8
$\mathcal{X} \sim \mathcal{D}(n, 10)$		**1.5517**	**1.5628**	**0.9897**	**92.2**	**1.2426**	**1.2436**	**0.9940**	**93.0**
Full-data	Sokoban	9.2978	10.4147	0.9594	60.92	14.8513	16.1940	0.9645	61.45
$\mathcal{X} \sim \mathcal{U}(n)$		7.1347	7.4233	0.9607	**61.62**	12.4740	**14.7325**	0.9500	48.92
$\mathcal{X} \sim \mathcal{D}(n, 10)$		**7.8141**	**8.0857**	**0.9614**	59.86	**13.3144**	12.4565	**0.9558**	**52.53**
Full-data	STP	4.3889	4.9981	0.9732	70.2	1.4297	1.6507	0.9353	57.0
$\mathcal{X} \sim \mathcal{U}(n)$		3.1497	**3.8005**	0.9633	61.2	1.0486	1.3083	**0.9404**	**69.0**
$\mathcal{X} \sim \mathcal{D}(n, 3)$		**3.1795**	3.7610	**0.9662**	**63.4**	**1.0917**	**1.5482**	0.9331	56.2

Table 10: Experimental results with $\mathcal{L}_{LM}$ by sampling from the $\mathcal{D}(n, \tau)$ distribution. Best scores are in **bold**.

Research Field	Relevance	Related Works with Summary
Learning Heuristics for Planning	In this work, we make use of previous methods to learn heuristics for planning. While These primarily studied neural architectures for this problem, we fix the architecture to an LM and study the data requirements.	**Machine Learning Perspective:** These works discuss classical ML techniques to learn heuristics (Yoon et al., 2006; Fern et al., 2011; Arfaee et al., 2011; ús Virseda et al., 2013; Chrestien et al., 2021; Groshev et al., 2018; Kirilenko et al., 2023). **Planner Perspective:** These incorporate planner properties to learn heuristics.(Yonetani et al., 2021; Vlastelica et al., 2019; Speck et al., 2021; Orseau et al., 2023; Orseau and Lelis, 2021; Kirilenko et al., 2023; Ernandes et al., 2004)
Heuristics with LMs	The previous works studied learning heuristics with classical machine learning techniques, here we specifically discuss how LMs are used in heuristic learning.	**Tree-Search in LLMs:** These discuss how various algorithms like DFS, BFS, MCTS can be combined with LLMs for planning (Yao et al., 2024; Hao et al., 2023; Chen et al., 2024). **LLMs with external planners:** These discuss how symbolic solvers can be augmented with LLMs. (Valmeekam et al., 2023; Gerevini et al., 2002; Liu et al., 2023; Yang et al., 2023; Guan et al., 2023; Dagan et al., 2023) **Improving LM-based heuristics:** These discuss how LM heuristics can be improved via training or prompting(Shinn et al., 2024; Zhou et al., 2023a; Lehnert et al., 2024; Gandhi et al., 2024).
Optimising Training Data	This is our problem statement for the planning task.	**Coreset Selection:** These works discuss the data requirements for training LMs, albeit for different tasks. To the best of our knowledge, we are the first to study coreset selection for planning (Paul et al., 2021; Marion et al., 2023; Abbas et al., 2023; Zhou et al., 2023b; Sorscher et al., 2022).

Table 11: A tabular summary of the related works discussed in Section 2.