AlphaNAS for LLMs: A Theoretical Framework for Optimizing Large Language Models using AlphaZero Methodology

Viraat Das
InferentIO Labs
viraat@inferent.io

Abstract

This paper introduces AlphaNAS, a theoretical framework for efficiently discovering high-performing and computationally economical architectures for large language models. Combining Monte Carlo Tree Search (MCTS) and a surrogate model, AlphaNAS efficiently explores the vast search space of large language models. By iteratively refining the surrogate model with newly explored architectures, AlphaNAS accurately estimates architecture performance and computational cost, enabling more informed decision-making during the search process. The presented work remains theoretical, and empirical evaluation will be a critical step for future studies. With successful validation, AlphaNAS could significantly influence the discovery of efficient architectures, thereby reducing the computational footprint of Large Language Models (LLMs).

***K*eywords** NAS · AlphaZero · LLM

1 Introduction

The advent of large language models (LLMs) has played a transformative role in numerous fields, including but not limited to, translation services, question-answering systems, and digital personal assistants. These models have rapidly emerged as one of the most significant breakthroughs in natural language processing. However, their resource-intensive nature poses challenges to their widespread adoption and deployment on edge and mobile devices, which have significant computational limitations.

Addressing this computational cost is of paramount importance as LLMs continue to be integral to many critical applications, with several organizations aiming to provide fine-tuned foundational models on a variety of edge devices. This aligns with the broadening goal of AI democratization, which seeks to make high-performing models more accessible to a larger population. A recent step in this direction has been the release of LLaMA by Meta AI [1], a state-of-the-art large language model. Designed to be more performant and smaller in size, LLaMA allows researchers and developers with limited infrastructure access to study and leverage these potent models.

Continuing in this vein, the LLaMA-Adapter approach [2] proposes a lightweight adaptation method for efficiently fine-tuning LLaMA into an instruction-following model. This method demonstrates a successful case of maintaining high model performance while introducing only a minimal number of learnable parameters, thereby reducing computational resources needed. This approach highlights the promise of adaptability in large language models, and points towards a direction of making these models more accessible and efficient.

In line with these efforts, this paper introduces AlphaNAS. This theoretically-grounded framework aims to discover high-performing and computationally economical architectures for LLMs. Combining Monte Carlo Tree Search (MCTS) and a surrogate model, AlphaNAS navigates the vast search space of LLMs more efficiently. The framework refines the surrogate model iteratively based on newly explored architectures, allowing it to accurately estimate architecture performance and computational cost, thereby facilitating informed decision-making during the search process.

While this work remains theoretical, its potential contributions to natural language generation tasks and the broader AI ecosystem are significant. Future research directions include empirical evaluation of AlphaNAS, exploration of variations of MCTS and alternate surrogate models, hardware-specific compilation, and further improvements. With successful validation and implementation, the proposed AlphaNAS framework could pave the way for resource-efficient deployment of large language models, thereby contributing to the broader objectives of AI accessibility and democratization.

2 Related Works

AlphaNAS draws inspiration from various research areas related to neural architecture search and optimization for large language models (LLMs). We discuss key works that have contributed to the development of AlphaNAS and highlight their relevance to its methodology.

Neural Architecture Search with Monte Carlo Tree Search (MCTS): MCTS has been widely explored in the domain of neural architecture search, and both AlphaNAS and AlphaX incorporate it as a fundamental component. Prior works such as DeepArchitect by Negrinho and Gordon (2017) and Wistuba (2017) employed vanilla MCTS for architecture exploration. In comparison, AlphaNAS distinguishes itself from AlphaX by focusing on efficient architecture exploration for LLMs without incorporating a meta-Deep Neural Network (DNN) like AlphaX [3, 4]. Instead, AlphaNAS introduces its own strategies and optimizations, such as surrogate models and iterative updates, to enhance the search process and discover LLM architectures that balance performance and computational efficiency.

AlphaZero and Reinforcement Learning: The influential work of AlphaZero by Silver et al. [5] in game-playing agents has inspired AlphaNAS. AlphaZero demonstrated the power of combining reinforcement learning with tree search algorithms, achieving state-of-the-art performance in various board games. AlphaNAS adopts a similar strategy, integrating MCTS with surrogate models [3, 4]. While AlphaZero focused on game-specific architectures, AlphaNAS extends this concept to the domain of LLMs, applying it to the search for optimal language model architectures.

Surrogate Models: Surrogate models in AlphaNAS provide efficient estimates of architecture performance and computational cost, serving as approximations of the true objective function [**?**]. Trained using evaluated architectures and their performance metrics, these models capture patterns and relationships between architectural configurations and outcomes. By leveraging surrogate models, AlphaNAS reduces computational burden, enables more efficient exploration of the architecture space, and leverages knowledge from previously evaluated architectures [3, 4]. Iterative updates refine the surrogate models, improving their predictive capabilities and search efficiency. The use of surrogate models in AlphaNAS facilitates the discovery of high-performing models in the complex search space of LLM architectures.

3 Overview of AlphaNAS for LLMs

3.1 Problem Definition

The primary goal of AlphaNAS is to search for an optimal architecture that minimizes computational cost while maintaining or improving the performance of LLMs. This can be mathematically expressed as follows:

$$\underset{a \in \mathcal{A}}{\operatorname{argmin}}[f(a) + g(a)] \tag{1}$$

where:

- $\mathcal{A}$ is the search space of LLM architectures
- $f(a)$ is the estimated computational cost of architecture a
- $g(a)$ is a measure of the model's performance such as perplexity or negative log-likelihood

3.2 High-level Procedure

AlphaNAS leverages a combination of MCTS and surrogate models to effectively explore the architecture space for LLMs. It maintains a balance between exploration of new architectures and exploitation of known architectures.

To address this problem, AlphaNAS combines two key components: Monte Carlo Tree Search (MCTS) and surrogate models. MCTS enables efficient exploration and exploitation of the architecture space, while surrogate models provide quick estimates of architecture performance and computational cost. This combination allows for effective navigation of the search space and the discovery of high-performing and computationally efficient architectures for LLMs.

The high-level overview of AlphaNAS is summarized in Algorithm 1. At each iteration of the MCTS, the algorithm selects an architecture a_i using the current surrogate model and evaluates it to obtain its performance metric y_i and computational cost c_i. The dataset $\mathcal{D}$ is updated with the new architecture and its associated metrics. The surrogate model is then retrained on the updated dataset, improving its predictions. This iterative process continues for multiple MCTS iterations, gradually refining the surrogate model and guiding the search towards optimal architectures.

Algorithm 1 AlphaNAS Overview

1: **Initialize:** Surrogate model $h(a; \Theta)$, dataset $\mathcal{D} \leftarrow \emptyset$
2: **for** each MCTS iteration i **do**
3: Run MCTS with current surrogate model to select architecture a_i
4: Evaluate a_i to obtain performance metric y_i and computational cost c_i
5: Update $\mathcal{D}$: $\mathcal{D} \leftarrow \mathcal{D} \cup (a_i, y_i, c_i)$
6: Retrain surrogate model on updated $\mathcal{D}$: $\Theta \leftarrow \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta; \mathcal{D})$
7: **end for**
8: **Return:** Surrogate model $h(a; \Theta)$

By iteratively updating the surrogate model and exploring the architecture space with MCTS, AlphaNAS gradually improves its ability to discover architectures that strike a balance between computational efficiency and model performance for LLMs.

4 Monte Carlo Tree Search (MCTS) in AlphaNAS

The AlphaNAS system employs MCTS to traverse the search space $\mathcal{A}$ efficiently. MCTS is a tree-based search algorithm that balances exploration and exploitation to navigate complex search spaces effectively. The MCTS algorithm consists of four main stages: selection, expansion, simulation, and backpropagation. We start out by defining how to construct the Search Space.

4.1 Search Space Definition

The search space $\mathcal{A}$ consists of a wide variety of LLM architectures, represented as a Cartesian product of sets corresponding to different architectural components and their configurations:

$$\mathcal{A} = \mathcal{L} \times \mathcal{T} \times \mathcal{C} \tag{2}$$

where

- $\mathcal{L}$ is the set of potential layer counts
- $\mathcal{T}$ is the set of layer types (e.g., transformer, LSTM, GRU)
- $\mathcal{C}$ is the set of configurations within layers (e.g., attention head counts in transformer layers)

Each element $a \in \mathcal{A}$ denotes a specific LLM architecture, characterized by its layer count, layer type, and configurations within layers. By systematically exploring and evaluating various combinations of architectural choices, AlphaNAS enables the discovery of architectures that balance performance and computational efficiency.

4.2 Monte Carlo Tree Search (MCTS)

The AlphaNAS system employs MCTS to traverse the search space $\mathcal{A}$ efficiently. MCTS is a tree-based search algorithm that balances exploration and exploitation to navigate complex search spaces effectively. The MCTS algorithm consists of four main stages: selection, expansion, simulation, and backpropagation.

4.3 Definition of Terms

- $Q(s, a)$ is the average reward observed for action a in state s
- $N(s)$ is the number of times state s has been visited
- $N(s, a)$ is the number of times action a has been taken in state s
- c is a constant that controls the balance between exploration and exploitation
- $\mathcal{A}$ is the search space for the LLM arhictectures

4.3.1 Selection

Starting from the root node, the algorithm traverses the search tree based on the UCB1 (Upper Confidence Bound 1) formula:

$$a = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \left[Q(s,a) + c\sqrt{\frac{2\log(N(s))}{N(s,a)}} \right] \tag{3}$$

In the selection stage, the AlphaNAS system traverses the search tree from the root node to a leaf node. At each node, an action corresponding to a potential modification in the architecture is selected. This selection is based on the UCB1 formula, which balances exploration (trying out less-visited actions) and exploitation (choosing actions with high expected rewards).

For example, consider a node representing an architecture with two Transformer layers. The potential actions at this node might include adding another layer, changing a Transformer layer to an LSTM layer, or adjusting the number of attention heads in one of the Transformer layers. AlphaNAS would select the action that has the highest UCB1 score, reflecting either a high expected reward or a high potential for exploration.

4.3.2 Expansion

Upon reaching a leaf node, the expansion stage occurs, where new child nodes representing modified architectures are added to the search tree. The specific transformations depend on the available actions defined by the search space.

During the expansion stage, AlphaNAS adds new child nodes to the search tree, representing architectures modified according to the action chosen during the selection stage.

Continuing from the example above, if the chosen action was to add another layer, the new node added during the expansion stage would represent an architecture with three Transformer layers. If the action was to change a Transformer layer to an LSTM layer, the new node would represent an architecture with one Transformer layer and one LSTM layer.

Once a leaf node s is reached, new child nodes are added to the tree. Each child node corresponds to an action $a \in \mathcal{A}(s)$ that could be taken in the current state.

4.3.3 Simulation

The simulation stage involves the estimation of the newly added architectures' performance. Due to the high computational cost of fully training and evaluating each possible architecture, AlphaNAS uses a surrogate model to quickly estimate the architecture's performance. The surrogate model is further explained in Section 5.

For instance, the surrogate model might estimate the performance of the new architecture with three Transformer layers to be a perplexity of 20 on a validation set. Alternatively, for the architecture with one Transformer layer and one LSTM layer, the surrogate model might estimate a higher perplexity of 25.

The simulation stage involves the evaluation of newly expanded nodes. Specifically, a reward r is estimated for each new child node s' using a surrogate model, denoted as $h(a;\Theta)$. The surrogate model estimates the performance of architecture a parameterized by Θ.

$$r = h(s';\Theta) \tag{4}$$

This reward r is an estimation of the model's performance, such as its perplexity or negative log-likelihood.

4.3.4 Backpropagation

Finally, during the backpropagation stage, the statistics and rewards estimated during the simulation are propagated back up the tree to the root node. The values of all visited nodes and edges in the tree are updated, allowing the system to learn from the simulations.

In our ongoing example, the new estimated performances (perplexities) of the child nodes would be backpropagated up to the parent node. This would update the parent node's understanding of the expected rewards associated with each action, informing future selection stages.

The rewards and statistics obtained from the simulation stage are used to update the values of the nodes and edges traversed during the selection stage. Specifically, for each visited state-action pair (s, a), the visit count $N(s, a)$ is incremented and the average reward $Q(s, a)$ is updated as follows:

$$N(s, a) \leftarrow N(s, a) + 1$$
$$Q(s, a) \leftarrow Q(s, a) + \frac{r - Q(s, a)}{N(s, a)}$$

Here, r is the reward estimated during the simulation stage.

This backpropagation process ensures that the estimated rewards and visit counts of the state-action pairs are kept up to date, which guides the selection of actions in future iterations of the search.

5 Surrogate Model

The surrogate model plays an essential role within AlphaNAS, allowing for a streamlined search within the architecture space without the need for exhaustive training and evaluation of every architecture.

5.1 Definition of Terms and Variables

- a: This represents a specific architecture drawn from the complete architecture space, denoted as $\mathcal{A}$.
- $h(a; \Theta)$: This function signifies the surrogate model's estimate of the performance for a given architecture a, influenced by the current state of its parameters Θ.
- y: This variable captures the actual performance of the architecture a, as derived from thorough training and evaluation.
- $c(a; \Theta)$: This function stands for the surrogate model's estimate of the computational cost for the architecture a. Like the performance estimate, this also depends on the model's parameters Θ.
- c: This represents the genuine computational cost incurred while training and evaluating the architecture a.
- $\mathcal{D}$: This symbolizes the dataset compiled for this process. $\mathcal{D}$ houses a set of triples (a, y, c), each capturing a unique architecture, its real-world performance, and its actual computational cost.
- $\mathcal{A}$: This encapsulates the comprehensive architecture space encompassing all conceivable architectures that could be evaluated.
- λ: This stands for a hyperparameter utilized in the model's loss function. Its value determines the balance between the emphasis placed on the accuracy of the model's performance prediction and its computational cost prediction.

5.2 Surrogate Model Construction

The construction of the surrogate model involves two main steps: Initialization and Surrogate Model Training.

5.2.1 Initialization

In the initialization step, a random subset of architectures is sampled from the architecture space $\mathcal{A}$. For each selected architecture a, a full-scale training and evaluation process is performed, which results in obtaining an actual performance y and a computational cost c. The performance y can be calculated using a number of metrics, including accuracy, precision, recall, F1-score, or, for text generation tasks, metrics such as the G-Eval framework.

The computational cost c is calculated considering both the training phase, which includes the total number of Floating Point Operations (FLOPs) required for training the architecture on a specific dataset and memory usage during training, and the inference phase, which includes the total FLOPs required for a single forward pass during the prediction phase after training and the memory used during inference.

These results (a, y, c) are then encapsulated into a triple and stored in the dataset $\mathcal{D}$, providing the initial training data for the surrogate model. In this triple, a represents an architecture, y is the actual performance of a, and c is the actual computational cost of a.

5.2.2 Surrogate Model Training

After obtaining the dataset $\mathcal{D}$, the surrogate model parameters Θ are trained by minimizing the loss function $\mathcal{L}$, defined as follows:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta; \mathcal{D}) \tag{5}$$

$$\mathcal{L}(\Theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(a,y,c)\in\mathcal{D}} \left[\lambda(h(a;\Theta) - y)^2 + (1-\lambda)(c(a;\Theta) - c)^2\right] \tag{6}$$

1. The surrogate model $h(a; \Theta)$ generates an estimate of the architecture's performance.
2. $c(a; \Theta)$ estimates its computational cost.
3. The loss function $\mathcal{L}$ minimizes the difference between the surrogate model's predictions and the actual values (y, c), for each architecture triple in $\mathcal{D}$
4. λ is a hyperparameter balancing the trade-off between performance prediction and computational cost prediction.

5.2.3 Iterative Update of the Surrogate Model

Once the surrogate model has been initialized and trained, the following iterative process is performed:

Step 1: Exploration

The surrogate model is utilized to guide the exploration of the architecture space $\mathcal{A}$. It predicts the performance and computational cost of unseen architectures, and based on these predictions, a potentially promising architecture a_i is selected for evaluation.

Step 2: Evaluation of Selected Architecture

The selected architecture a_i is fully trained and evaluated, obtaining its actual performance y_i and computational cost c_i. These results are encapsulated into a new triple (a_i, y_i, c_i) and added to the dataset $\mathcal{D}$.

Step 3: Surrogate Model Update

The surrogate model parameters Θ are updated by re-optimizing the loss function $\mathcal{L}$ using the updated dataset $\mathcal{D}$.

$$\Theta \leftarrow \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta; \mathcal{D}) \tag{7}$$

These steps are repeated, iteratively refining the surrogate model's predictive power and guiding the search for architectures through the architecture space $\mathcal{A}$. Over time, this iterative process should guide the model towards increasingly effective and efficient architectures for the given task.

5.3 Final Architecture Selection

The final architecture, a^*, is selected to maximize the estimated performance while adhering to computational constraints:

$$a^* = \underset{a\in\mathcal{A}}{\operatorname{argmax}} [h(a;\Theta) - \lambda c(a;\Theta)] \tag{8}$$

Here, λ is used again to balance the trade-off between performance and computational cost.

5.4 Summary

This is the procedure summarized:

Algorithm 2 AlphaNAS Surrogate Model Construction and Training

Initialize dataset $\mathcal{D} \leftarrow \emptyset$
for each a in a set of randomly sampled architectures from $\mathcal{A}$ **do**
    Train and evaluate a, obtaining performance y and cost c
    Add the triple (a, y, c) to $\mathcal{D}$
end for
Train surrogate model $h(a; \Theta)$ using $\mathcal{D}$, optimizing Θ by minimizing the loss function $\mathcal{L}$
while architecture search is not complete **do**
    Use surrogate model to select a potentially promising architecture a_i from $\mathcal{A}$
    Train and evaluate a_i, obtaining performance y_i and cost c_i
    Add the triple (a_i, y_i, c_i) to $\mathcal{D}$
    Re-train the surrogate model using the updated $\mathcal{D}$
end while
Select the final architecture a^* as $a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}[h(a; \Theta) - \lambda c(a; \Theta)]$

6 Future Works

This paper lays the theoretical groundwork for utilizing Monte Carlo Tree Search (MCTS) and surrogate models to uncover efficient architectures for large language models. To advance this groundwork and substantiate its practicality, there are several possible paths for future exploration:

6.1 Empirical Validation

Implementation of the outlined AlphaNAS framework, and proceeding with rigorous experiments, forms the next necessary step in the empirical validation of its potential. This includes a comprehensive evaluation against existing architecture search methods, benchmarked across various metrics and datasets, thus offering a thorough examination of its efficacy and efficiency in identifying performant and cost-effective architectures for large language models.

6.2 Integration of Hardware-Aware NAS

The integration of a hardware-conscious model within the optimization framework could act as a form of low-level optimization. Inspired by the research of Cummings et al. [6], which proposed a strategy to expedite the neural architecture search process by acknowledging the specific attributes of the intended hardware, similar incorporations could serve to enhance AlphaNAS. Such an approach would aid in the identification of architectures that deliver superior performance while also maintaining compatibility with the specific computational constraints of the hardware environment intended for deployment.

6.3 Expansion Towards Pruned Architectures

Another direction to consider is extending the AlphaNAS framework to account for pruned architectures. Building on the work of Frantar and Alistarh [7], who exhibited that large language models could be pruned in a one-shot method while maintaining their accuracy, AlphaNAS could integrate this concept. Consequently, this would enable the framework to discover both the original and pruned architecture variations, potentially leading to architectures that are more efficient and performance-optimized.

7 Conclusion

AlphaNAS is a proposed framework that combines Monte Carlo Tree Search (MCTS) and a surrogate model to efficiently explore the search space of large language models. By continually updating the surrogate model based on newly explored architectures, AlphaNAS allows for efficient estimates of architecture performance and computational cost, facilitating informed decisions during the search process. While our discussion has primarily focused on the theoretical aspects, empirical evaluation remains an important future direction to validate the effectiveness and practical applicability of AlphaNAS. The proposed framework lays the groundwork for a potential research direction for discovering high-performing and computationally efficient architectures for large language models, contributing to advancements in natural language generation tasks.

References

[1] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[2] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention, 2023.

[3] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search, 2019.

[4] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search, 2019.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[6] Daniel Cummings, Anthony Sarah, Sharath Nittur Sridhar, Maciej Szankin, Juan Pablo Munoz, and Sairam Sundaresan. A hardware-aware framework for accelerating neural architecture search across modalities, 2022.

[7] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023.