PUZZLE: DISTILLATION-BASED NAS FOR INFERENCE-OPTIMIZED LLMS

Akhiad Bercovich*, Tomer Ronen*, Talor Abramovich, Nir Ailon, Nave Assaf, Mohammad Dabbah, Ido Galil, Amnon Geifman, Yonatan Geifman, Izhak Golan, Netanel Haber, Ehud Karpas, Itay Levy, Shahar Mor, Zach Moshe, Najeeb Nabwani, Omri Puny, Ran Rubin, Itamar Schen, Ido Shahaf, Oren Tropp, Omer Ullman Argov, Ran Zilberstein, and Ran El-Yaniv

NVIDIA

{abercovich, tronen, relyaniv}@nvidia.com

ABSTRACT

Large language models (LLMs) have demonstrated remarkable capabilities, but their adoption is limited by high computational costs during inference. While increasing parameter counts enhances accuracy, it also widens the gap between state-of-the-art capabilities and practical deployability. We present *Puzzle*, a framework to accelerate LLM inference on specific hardware while preserving their capabilities. Through an innovative application of neural architecture search (NAS) at an unprecedented scale, Puzzle systematically optimizes models with tens of billions of parameters under hardware constraints. Our approach utilizes blockwise local knowledge distillation (BLD) for parallel architecture exploration and employs mixed-integer programming for precise constraint optimization.

We demonstrate the real-world impact of our framework through Llama-3.1-Nemotron-51B-Instruct (Nemotron-51B), a publicly available model derived from Llama-3.1-70B-Instruct. Nemotron-51B achieves a 2.17x inference throughput speedup, fitting on a single NVIDIA H100 GPU while preserving 98.4% of the original model's capabilities. Nemotron-51B currently stands as the most accurate language model capable of inference on a single GPU with large batch sizes. Remarkably, this transformation required just 45B training tokens, compared to over 15T tokens used for the 70B model it was derived from. This establishes a new paradigm where powerful models can be optimized for efficient deployment with only negligible compromise of their capabilities, demonstrating that inference performance, not parameter count alone, should guide model selection. With the release of Nemotron-51B and the presentation of the Puzzle framework, we provide practitioners immediate access to state-of-the-art language modeling capabilities at significantly reduced computational costs.

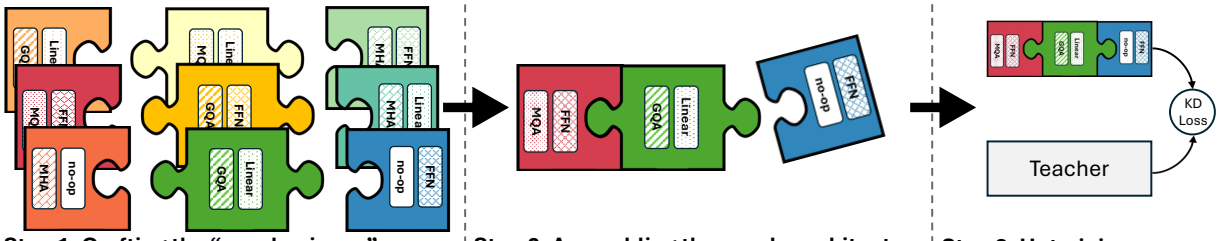
1 Introduction

With remarkable advancements in the capability and accuracy of LLMs, these models are increasingly adopted across various domains, ranging from virtual assistants to sophisticated enterprise solutions. This adoption trend is accompanied by a growing appetite for larger and more powerful models, driven in part by the aspirational goal of achieving artificial general intelligence (AGI), as evidenced by the industry's push toward increasingly larger-scale LLMs [39, 6, 1, 13] and inference-time complexity (GPT-o1, [47, 51, 16]). However, the high computational costs associated with these models and their projected future iterations – particularly during inference – restrict their accessibility and scalability, thus presenting a significant challenge for widespread commercial applications.

LLMs require a substantial amount of parameters for their training process to converge easily and achieve better generalization [26, 22, 5, 8]. This overparameterization not only facilitates optimization, but also provides greater capacity to store knowledge and learn complex patterns across diverse tasks, explaining why larger models consistently

*These authors contributed equally. Other co-authors are listed alphabetically.

demonstrate superior performance [26, 22]. However, once trained, many parameters and computations turn out to be redundant for inference, as evidenced by the success of various computational efficiency techniques [18, 9, 50, 7, 34, 24, 3]. Yet, LLM architectures remain largely uniform, comprising repeated identical layers, with little consideration given to balancing each block's computational cost against its contribution to overall model predictive performance—a design choice primarily driven by training stability and ease of scaling rather than inference efficiency. This work addresses how to transform a trained LLM from a structure suited for training into one optimized for efficient inference on specific hardware (such as H100), while preserving its accumulated knowledge and predictive performance. Given a "*parent model*", our approach explores a large search space of architecture configurations to identify efficient options tailored to meet specific hardware and task-related constraints. This exploration requires a method to reliably estimate the performance of each potential configuration, allowing us to identify models that balance efficiency and accuracy for deployment.



Step 1: Crafting the "puzzle pieces"
Applying block-wise local distillation to every alternative subblock replacement in parallel and scoring its quality and inference cost to build a "library" of blocks.

Step 2: Assembling the puzzle architecture
Utilizing Mixed-Integer-Programming to assemble a heterogeneous architecture that optimizes quality under constraints such as throughput, latency and memory usage.

Step 3: Uptraining
The reassembled architecture is trained with global Knowledge-Distillation to strengthen inter-block compatibility.

Figure 1: An overview of the three stages of our Puzzle framework.

In this paper we introduce the *Puzzle* framework, summarized in Figure 1, which pioneers a decomposed *neural architecture search* (NAS) strategy for LLMs to explore and optimize a vast search space of possible model architectures for target hardware. Inspired by methods in computer vision NAS, and LANA [36] especially, we define design spaces that include alternative attention and feed-forward network (FFN) layers of varying efficiency degrees, up to a complete layer skip in the extreme case. We then use a *blockwise local distillation* (BLD) (See Section 3) framework to train all these block variants for all layers of the parent LLM in parallel. Next, we efficiently score each alternative replacement "puzzle piece" and search an enormous design space for the most accurate models, while adhering to a set of inference constraints, such as memory size, latency, and throughput. This is done by utilizing a *Mixed-Integer-Programming* (MIP) algorithm. Lastly, the reassembled model is trained with *Global Knowledge Distillation* (GKD) [20]. Unlike traditional uniform transformer architectures, our NAS framework produces non-uniform models with adapted computation allocation, optimizing each layer's expressivity based on the model's overall requirements to focus resources where they matter most. This leads to significant gains in efficiency without compromising model expressivity. By focusing on parent models with SOTA performance, we derive child models pushing the efficient frontier (see Figure 5 and Table 5), e.g., models which provide the best accuracy per dollar.

Our framework offers several advantages. First, it enjoys extremely low costs relative to training a model from scratch. For instance, the entire training—BLD before the MIP stage and GKD afterward—required less than 50B tokens to run on Llama-3.1-70B-Instruct, compared to more than 15T tokens used to train the parent model. Additionally, our method requires only the parent model's weights—not its training data—making it ideal for "open-weights, closed-data" scenarios where training data of the parent model is not publicly available. This allows practitioners to take any freely-available model and tailor it to their specific hardware or use case. To demonstrate the effectiveness of our framework, we present Llama-3.1-Nemotron-51B-Instruct (Nemotron-51B), derived from the Llama-3.1-70B-Instruct parent model using Puzzle. Nemotron-51B breaks the efficient frontier of LLMs on a single NVIDIA H100 GPU, establishing a new state-of-the-art for commercial applications, by achieving unmatched throughput and memory efficiency on this hardware. Interestingly, the Nemotron-51B resulting architecture is unique and irregular, with many layers featuring reduced or skipped attention and FFN operations. This design enhances NVIDIA H100 utilization under FP8 quantization while preserving accuracy. We also introduce a derivative of Llama-3.1-8B-Instruct, which breaks the efficient frontier for its throughput slice. While Nemotron-51B also leads within its parameter range, we argue that categorizing models solely by parameter count—such as 50B or 70B—is inadequate for real-world applications. Instead, inference performance under specific hardware, inference engine, quantization levels, budget constraints, and usage profiles—such as varying sequence lengths and batch sizes—should guide model selection.

Conventional approaches to designing LLMs for inference, such as training from scratch or knowledge distillation, present significant challenges. Training a model from scratch is slow and resource-intensive, making it impractical for evaluating multiple configurations. Knowledge distillation, while generally faster due to guidance from a teacher model, remains prohibitively costly when evaluating multiple candidates from a large search space. Puzzle circumvents these limitations by conducting architecture search immediately after BLD, during the MIP stage (Stage 2 in Figure 1). These stages efficiently identify promising configurations for multiple levels (slices) of inference constraints, without requiring full-model KD for each candidate. The computationally intensive GKD process is reserved for the final stage, after the optimal architectures have been reassembled, enabling Puzzle to focus resources on refining a single optimized model for each slice while keeping the overall cost low.

Our contributions:

(1) We introduce Puzzle, a framework that applies decomposed neural architecture search to distill an LLM into hardware and inference scenario optimized models. Our framework flexibly supports optimization across ranges of multiple constraints, including throughput, latency and memory usage. Our work pioneers the large-scale use of blockwise distillation and MIP-based architecture search for LLMs, successfully scaling these techniques to tens of billions of parameters while requiring only a fraction of the original training compute. Puzzle is designed to be low-cost, enabling the efficient creation of multiple child models from a single parent LLM—each tailored to different hardware and inference requirements. This scalability makes it feasible to publish optimized variants for diverse use cases and deployment environments.

(2) Using Puzzle, we introduce Llama-3.1-Nemotron-51B-Instruct, a model optimized for the NVIDIA H100 GPU thus setting a new benchmark for throughput and memory efficiency in commercial applications. Our results demonstrate that hardware-specific architecture optimization can achieve superior efficiency compared to traditional uniform architectures while maintaining model performance.

(3) Our method focuses on optimization for real world scenarios, running on actual inference engines and efficient quantization levels (FP8). We therefore enhance TensorRT-LLM to efficiently support non-uniform blocks and attention mechanisms with varying numbers of key-value heads across layers. This results in models that can directly improve the costs and usability of running LLM inference in practical applications (see Section 6).

(4) We provide a comprehensive empirical analysis of the relationship between architectural choices and hardware efficiency, offering insights to guide the design of future hardware-aware LLM architectures.

These contributions advance the field of LLM optimization by transforming powerful language models into efficient, deployment-ready systems while preserving their capabilities. The open-source release of Nemotron-51B demonstrates that state-of-the-art language models can run efficiently on standard hardware, making powerful AI accessible with modest computing resources. This work not only opens new directions for automated architecture optimization but also represents a significant step toward democratizing access to advanced AI technology.

2 Search Space

The motivation for applying NAS in our work lies in the redundancy found in many trained LLMs. Numerous studies have shown that a significant portion of computations and parameters in LLMs become redundant post-training, during inference [44, 24, 3]. Prior studies tried to solve these issues with techniques such as pruning [38, 35, 7, 34, 49, 48], removing entire layers [18], local attention methods (e.g., window and sink attention) [9, 50], reducing the number of key-value (KV) heads [45, 4] and many more. Given the high computational and monetary cost associated with running LLMs, optimizing these models to eliminate inefficiencies becomes a critical goal. NAS methods are defined by the search space, the search strategy and evaluation metric of candidate architectures. NAS offers a systematic approach to exploring architectural changes that balances performance and resource constraints, and thus is a prime candidate for reducing inference costs.

In this work, we defined a vast search space, encompassing different operations for each layer of the parent LLM model. A *block* is composed of smaller components called *subblocks*. While blocks are user-defined, in LLMs, a block typically refers to a transformer layer, with the subblocks being the attention module and the feed-forward network (FFN). For each transformer layer i, the search space combines options for the attention subblock (denoted $\mathcal{A}_i = \{a_j\}_{j=1}^{m}$, where m is the number of possible attention subblocks) and FFN subblock (denoted $\mathcal{F}_i = \{f_k\}_{k=1}^{n}$). The attention subblock options could include mechanisms such as standard multi-head attention (MHA), *grouped query attention* (GQA) [4] with varying numbers of key-value heads, replacing the attention layer with a single linear layer, or no-op (i.e., entirely skipping the subblock). The FFN options include full or reduced intermediate dimensions (which could be obtained with pruning techniques), linear layers, or no-ops. The combined search space for each transformer layer (or parent block), represented as $\mathcal{A}_i \times \mathcal{F}_i$, captures all possible pairings of attention and FFN

configurations. In this work, each parent transformer layer is replaced by a single corresponding child transformer layer (block), although theoretically, multiple parent layers could be grouped and replaced by a different number of child blocks (i.e., P parent layers to L child blocks). Specifically:

- **Attention subblocks**: For $\mathcal{A}_i$, we consider different variants of GQA with varying numbers of key-value heads (8, 4, 2, and 1). We also include the option to replace the entire attention subblock with a single linear layer or skip it entirely using a no-op operation.
- **FFN subblocks**: For $\mathcal{F}_i$, we consider the full intermediate-dimension expansion factor of the parent model, along with reduced dimensions of approximately: 87%, 75%, 50%, 25%, 20% and 10% of the original intermediate size. Furthermore, linear layers and no-op options are also included.

These variants offer a tradeoff between memory efficiency, computational cost, and representational power, allowing for flexibility based on specific resource constraints and performance needs. For example, reducing the number of key-value heads (by using GQA with fewer heads) not only speeds up the attention computation, but also helps decrease memory usage by reducing the KV-cache size, which can be crucial for meeting memory constraints or enabling larger batch sizes for better throughput (as GPUs operate more efficiently with larger batches). Using linear layers or reduced FFN dimensions can similarly lower computational requirements, whereas keeping full dimensions maintains higher representational power for better accuracy. Lastly, in this work, we require that all subblocks within a layer have the same input and output dimensions as their parent. However, a scheme with subblocks of varying embedding dimensions could be developed in future work.

To illustrate the scale of the search space, consider Llama 3.1-70B [13], a model with 80 transformer layers. For the specific instantiation of our framework presented in this work, we defined each transformer layer to have 6 potential alternatives for the attention subblock and 9 alternatives for the FFN subblock, resulting in 54 possible configurations per layer. Consequently, the total number of potential child model architectures is 54^{80}, which is approximately 10^{138} different architectures. This number greatly exceeds the estimated number of atoms in the observable universe (10^{82}). Given such an immense number of possibilities, and considering the costs of partially training or even measuring the accuracy of a single LLM architecture, evaluating any representative subset of the search space is computationally infeasible. Therefore, designing a traditional search strategy and evaluation scheme for NAS in such a vast space is challenging. To address this, we devised an efficient decomposed local distillation and evaluation framework, and our decomposed search strategy (described in Section 3 and Section 4). These strategies allow feasible navigation of the search space to find configurations that balance expressivity and efficiency with practical constraints like latency, memory usage, and throughput.

3 Blockwise Local Distillation

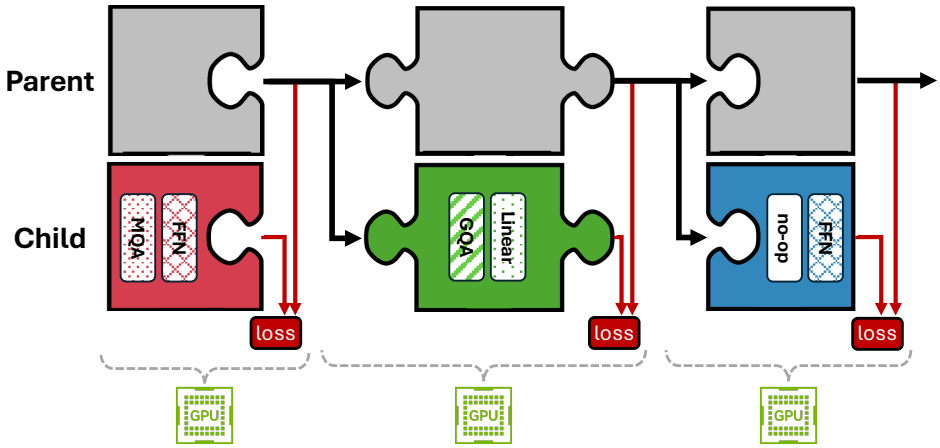


Figure 2: Blockwise local distillation (BLD): each block is trained in parallel and independently.

To create capable "puzzle pieces" —-a set of trained block variants forming a *block library* for architectural exploration—-we need to set effective weights for each child block. Our method involves efficient, training-free initializations for these blocks (see Section 3.2). While these initialization techniques are beneficial for low-budget experiments, performance can be significantly improved by distilling knowledge from each parent block to its corresponding child block.

Our approach is to decompose the crafting process to operate on individual blocks instead of complete child models, which drastically decreases the computational cost. We train each child block independently and in parallel to locally mimic its corresponding parent block, with only the parent activations being transferred between layers, as illustrated in Figure 2. This local distillation approach offers several advantages. First, because each child block relies solely on its corresponding parent block, activations and gradients are isolated from other child blocks. This independence enables training blocks separately, leveraging pipeline parallelism across multiple GPUs. Second, each child subblock is trained to mimic a relatively simple function—a single parent subblock—making the process considerably simpler and more stable than training an entire child model. This focused training facilitates faster convergence and allows higher learning rates compared to standard language modeling or GKD methods. Additionally, we find that this approach requires only a small dataset (approximately one billion tokens). Third, each child subblock benefits from high-quality outputs from its preceding parent subblock, rather than the lower-quality outputs typical in global model training, which further enhances convergence speed.

To optimize the performance of each child block, we feed parent activations into the current block and compute a normalized mean squared error (MSE) loss [27]. Specifically, we define the loss as $\mathcal{L} = \frac{\text{MSE}(o_p, o_c)}{\text{MSE}(o_p, 0)}$, where o_p and o_c represent the outputs of the original parent block and the modified child block, respectively.

A primary limitation of BLD is that it does not ensure compatibility between different blocks. This issue arises because each block is trained with inputs from the preceding parent blocks rather than from the outputs of its predecessor blocks within the child model. This prevents later blocks from adapting to the errors of earlier blocks, and may lead to errors propagating and compounding through the child model. To mitigate this, we introduce GKD as a final training phase in our framework (see Section 5). Nonetheless, empirical results show that **the BLD stage alone recovers much of the parent model's performance** (see Table 13).

To ensure broad coverage of diverse data domains within limited training schedules, we curated a dataset mixture, termed *Distillation Mix*, for all our distillation training runs. This mixture includes source code repositories, Wikipedia articles, books, news websites, and several other domains. The dataset comprises 224 billion tokens collected from three public datasets: FineWeb [40], Dolma [46], and Buzz-V1.2 [21]. Our BLD experiments use 1B training tokens. In our BLD experiments, we used 1 billion training tokens. We discuss the effect of varying BLD training lengths on downstream tasks in Section 8.1.3.

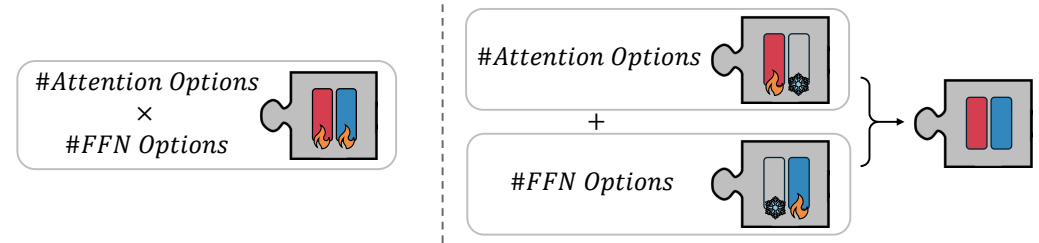
3.1 Building a Block Library with Decoupled Blockwise Local Distillation

The first stage of our decomposed NAS framework (further discussed in Section 4) is building a "library" of trained blocks. In order to cover the entire search space defined in Section 2, we need to obtain trained weights for each attention variant a_j and each FFN variant f_k in each transformer layer i. We consider two methods to train the block library: *coupled BLD* and *decoupled BLD*. For each transformer layer i, coupled BLD constructs each possible block variant $[a_j, f_k]_i$ and trains it to emulate the corresponding parent block $[a_{\text{parent}}, f_{\text{parent}}]_i$. This approach is similar to the aforementioned decomposed NAS methods used in CV. Given the significantly higher computational costs of LLMs compared to CV models, and noting the inherent structure of the transformer layer, we propose decoupled BLD to drastically reduce the cost of building a block library: training $[a_j, f_{\text{parent}}^{(\text{frozen})}]_i$ and $[a_{\text{parent}}^{(\text{frozen})}, f_k]_i$ separately to emulate $[a_{\text{parent}}, f_{\text{parent}}]_i$ while freezing the weights of the student subblock that is identical to the parent, and composing the trained subblocks into a full block $[a_j, f_k]_i$ only after training.

Given m attention variants, n FFN variants, and l transformer layers, coupled BLD requires training $m \cdot n \cdot l$ variants, while decoupled BLD requires only $(m + n) \cdot l$ variants, significantly speeding up library construction, visualized in Figure 3. This efficiency becomes especially critical with a large toolbox for $\mathcal{A}_i$ and $\mathcal{F}_i$. For instance, with $m = n = 20$ and $l = 80$ layers, decoupled BLD would require training 3,200 blocks compared to 32,000 for coupled BLD. Decoupled BLD enables us to explore a large search space and produce high quality models, without prior knowledge on which combinations of attention and FFN are most preferable. In Section 8.1.1 we present a technique to combine coupled BLD and decoupled BLD.

3.2 Initialization of Alternative Subblocks

To accelerate the distillation process, we introduce training-free initialization techniques for alternative subblocks. We propose a method to reduce the intermediate dimensions of FFN subblocks by selectively pruning channels based on their contribution. Our approach, called *Channel Contribution*, uses an activation-based strategy to estimate channel contribution during forward passes over a calibration dataset, similar to [38]. To guide pruning, we rank intermediate channels based on their impact on the FFN output. This is quantified as the distance between the original FFN output



Coupled training: train every block combination.

Decoupled training: train every subblock option together with a frozen parent subblock and then compose them to create an entire block. Block scores are computed on composed blocks, just as they are in coupled training.

Figure 3: Coupled BLD requires training $|\mathcal{A}_i| \times |\mathcal{F}_i|$ variants per transformer layer, while decoupled BLD requires only $|\mathcal{A}_i| + |\mathcal{F}_i|$ variants per layer, significantly speeding up library construction.

and the output after pruning. The channels with the lowest contributions are prioritized for pruning during subblock initialization.

To formalize our method, let us denote the hidden dimension by H, the FFN intermediate dimension by I, the FFN down projection matrix by $W^{\text{down}} \in \mathbb{R}^{I \times H}$. Let $X \in \mathbb{R}^I$ represent the FFN's intermediate activations for a single token. The output of the FFN, $Y \in \mathbb{R}^H$, is then given by

$$Y = (W^{\text{down}})^\top X = \sum_{k=1}^{I} X_k W^{\text{down}}_{k,:}$$

We define the per-token contribution of channel i as:

$$C_i(X) = \left\| \left(\sum_{k=1}^{I} X_k W^{\text{down}}_{k,:} \right) - \left(\sum_{\substack{k=1 \\ k \neq i}}^{I} X_k W^{\text{down}}_{k,:} \right) \right\|_2 = |X_i| \cdot \left\| W^{\text{down}}_{i,:} \right\|_2$$

We then compute the average contribution of each channel across all tokens in the calibration dataset.

When replacing an FFN subblock with a linear layer, we initialize it by computing the product of the up and down projection matrices for FFNs, thereby effectively ignoring the gating mechanism.

In attention subblocks with a reduced number of key-value heads, we initialize the projection matrices for the key and value heads by mean-pooling them into single projection matrices, following the approach used in [4]. When replacing an attention subblock with a linear layer, we use the product of the value and output projection matrices, which simulates the scenario where each token attends only to itself.

4 Decomposed NAS Search Algorithm for LLMs

At its core, our decomposed NAS framework is similar to earlier decomposed NAS methods used in computer vision (CV) such as DNA [30], DONNA [37], and especially LANA [36]:

1. **Build a block library:** construct a diverse block library using BLD (see Section 3.1).
2. **Estimate block resource requirements:** estimate the runtime and memory requirements of each block variant across different scenarios such as inference devices and sequence lengths.
3. **Score blocks:** score each block variant in each location inside the network to estimate its quality as an alternative to the parent block in that location.
4. **Search architectures:** use a search algorithm to construct "Puzzle architectures" that have the best estimated quality under specified runtime and memory constraints.

While these steps mirror established NAS approaches, their application to LLMs presents unique challenges due to the massive scale of the models. Our innovations in scaling these techniques to multi-billion-parameter models while maintaining efficiency are detailed in the following subsections.

4.1 Estimating Resource Requirements

Accurate estimation of computational resources is crucial for optimizing LLM architectures for real-world deployment. While theoretical metrics like FLOPs or parameter count are commonly used to approximate computational costs, they often fail to capture the complex realities of hardware acceleration. The actual runtime of neural network operations depends on numerous hardware-specific factors: the number of streaming multiprocessors (SMs), tensor cores, memory bandwidth, and I/O patterns all significantly impact performance. For instance, operations with fewer FLOPs might run slower in practice due to poor hardware utilization, particularly when processing small tensors. This disparity between theoretical and actual performance makes direct measurement on target hardware essential for meaningful optimization.

Memory requirements during inference comprise two distinct components with different scaling behaviors. The parameter memory, while substantial, remains constant regardless of the input size. In contrast, the key-value cache memory scales linearly with both batch size and sequence length, often becoming the dominant factor in long-sequence scenarios. For example, in a model with 32 attention heads and 128-dimensional head size, each token requires 8KB of KV-cache per layer using FP16 precision. For an 8K-token sequence with batch size 64, this amounts to 4GB per layer just for the KV-cache, potentially exceeding the parameter memory.

LLM inference operates in two distinct phases with markedly different performance characteristics. The prefill phase processes the initial input sequence in a single forward pass, enabling high parallelization. While efficient, the computational cost can become significant for long contexts. In contrast, the generation phase processes one query token at a time auto-regressively, requiring repeated forward passes, and often employing paged attention to manage memory efficiently. This difference between phases makes it crucial to actually measure both prefill and generation runtime in the target scenarios (examples measuring Nemotron-51B on several scenarios are shown in Table 3).

Batch size plays a particularly critical role in hardware efficiency, especially during the generation phase. During prefill, even small batches process substantial amounts of data due to sequence length, allowing efficient hardware utilization. However, generation with small batch sizes processes minimal data per layer while still performing all the IO needed to load the layer parameters, leading to severe hardware under-utilization. Increasing batch size creates larger tensors that better utilize GPU resources, often leading to significant improvements in throughput.

Given these complexities, our approach measures resource requirements directly on target hardware across various scenarios. For each block variant, we collect prefill and generation latencies across different sequence lengths and batch sizes at a chosen quantization level. These measurements directly determine the constraints in our MIP optimization (Section 4.3), enabling the search algorithm to find architectures that perform well in actual deployment rather than just in theory.

4.2 Scoring Architecture Solutions

A key advantage of decomposed NAS is its ability to estimate the quality of an assembled model based on metrics gathered from its individual blocks. This capability allows search algorithms to explore an enormous search space very efficiently, as the quality of each candidate architecture encountered during the search can be estimated within less than a second, instead of having to actually realize the candidate model and calculate some measure that requires performing forward passes on the entire model such as validation accuracy. Traditional NAS methods in CV, such as those in [55] and [42], typically rely on full model evaluation, which is computationally prohibitive for LLMs, where both search spaces and inference costs are substantially larger.

We score each block variant at each network location by measuring the impact of replacing only that specific block in the parent model. To do this, we construct a model identical to the parent but with a single block replaced by a trained block from our library, then calculate a performance measure on the entire model.For efficient I/O, when scoring multiple variants, we load onto the GPU only the blocks that differ from the previously evaluated model. We call these scores *replace-1-block scores*. During architecture search, we estimate the quality of a constructed architecture as the sum of its individual replace-1-block scores (see Section 4.3). Note that the scoring is not performed on candidate models during the search phase – we estimate the quality of each block only once, then use its replace-1-block score to estimate the quality of any candidate that contains it.

We consider several metrics as potential replace-1-block scores: *(1) downstream accuracy:* accuracy on downstream tasks such as MMLU [19]. While downstream accuracy is popular in Computer Vision applications, it can be very costly to measure in LLMs, especially given the number of block variants requiring scoring. *(2) LM loss:* causal language modeling loss, defined as the average log-likelihood of a validation corpus under the model's next-token prediction distribution. *(3) KL divergence:* Kullback–Leibler divergence between the next-token prediction distributions of the evaluated model and the parent model, averaged across tokens in a validation corpus. KL divergence is widely

used in KD setups as a statistical distance measure but has not been explored for decomposed NAS scoring until now. Our analysis in Section 8.1.4 highlights its effectiveness for block scoring.

4.3 Search Algorithm: Mixed-integer Programming

The search space for LLM architecture optimization is enormous (we consider $\sim 10^{138}$ possibilities), as discussed in Section 2. Efficiently navigating this space requires two key components: a fast method to estimate candidate quality, which we addressed in Section 4.2, and an efficient algorithm to maximize the estimated quality under hardware-specific constraints.

The structure of transformer models naturally frames our optimization problem as a grouped variant of the classical *Knapsack Problem*. Each layer of the model represents a group, containing various block alternatives (attention and FFN variants) as items. Each block alternative has an associated value (its quality score) and multiple costs (parameter memory, KV-cache memory, and runtime characteristics). The objective is to select exactly one block variant from each layer while maximizing the total score and satisfying deployment constraints.

Following [36], we formulate this as a Mixed Integer Programming (MIP) problem. Let $x_{i,j}$ be a binary decision variable indicating whether block variant j is selected for layer i. For a model with L layers and K_i variants per layer, the optimization problem is:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{L}\sum_{j=1}^{K_i} \mathsf{score}(i,j)x_{i,j} \\
\text{subject to} \quad & \sum_{i=1}^{L}\sum_{j=1}^{K_i} [\mathsf{mem}_{\mathsf{params}}(i,j) + b \cdot \mathsf{mem}_{\mathsf{kv}}(i,j)]x_{i,j} \leq \mathsf{Memory}_{\mathsf{max}} \\
& \frac{b \cdot \mathsf{seq\_len}}{\sum_{i=1}^{L}\sum_{j=1}^{K_i} \mathsf{runtime}(i,j,b)x_{i,j}} \geq \mathsf{Throughput}_{\mathsf{min}} \\
& \sum_{i=1}^{L}\sum_{j=1}^{K_i} \mathsf{runtime}(i,j,b)x_{i,j} \leq \mathsf{Latency}_{\mathsf{max}} \\
& \sum_{j=1}^{K_i} x_{i,j} = 1 \quad \forall i \in \{1,\ldots,L\} \\
& x_{i,j} \in \{0,1\} \quad \forall i,j \ ,
\end{aligned}
$$

where:

- $\mathsf{score}(i,j)$ is the quality score of block variant j in layer i, measuring how well it maintains the parent model's performance. If the block scores represent a negative impact (e.g. LM loss or KL divergence) we minimize the sum of scores instead of maximizing it.
- $\mathsf{mem}_{\mathsf{params}}(i,j)$ is the parameter memory required for block variant j in layer i, which is shared across all sequences in a batch.
- $\mathsf{mem}_{\mathsf{kv}}(i,j)$ is the key-value cache memory required for a single sequence in layer i with block variant j.
- b is the batch size - the number of sequences processed in parallel during inference.
- $\mathsf{seq\_len}$ is the total sequence length, including both prefill and generation.
- $\mathsf{runtime}(i,j,b)$ is the runtime of block variant j in layer i when processing batch size b.
- $\mathsf{Memory}_{\mathsf{max}}$ is the maximum allowed total memory, specified to fit the target GPU(s).
- $\mathsf{Throughput}_{\mathsf{min}}$ is the minimum required throughput (tokens per second).
- $\mathsf{Latency}_{\mathsf{max}}$ is the maximum allowed latency per batch.

The objective maximizes the sum of quality scores across all selected block variants. The first constraint ensures the total memory usage stays within limits, accounting for both parameter memory (shared across batches) and KV-cache memory (which scales linearly with batch size as each sequence requires its own cache). The second constraint enforces a minimum throughput requirement: for batch size b, we process $b \cdot \mathsf{seq\_len}$ tokens within the total runtime,

which must meet or exceed $\mathsf{Throughput}_{\mathsf{min}}$ tokens per second. The third constraint ensures the total processing time for a batch does not exceed the maximum allowed latency. The fourth constraint guarantees exactly one variant is selected for each layer. Note that we do not impose any constraint on the number of model parameters - they are reduced naturally by the search algorithm due to the true scenario constraints.

Since the batch size b is not a variable in this optimization, we solve the MIP problem multiple times with different values of b to explore the runtime-memory trade-off space. Larger batch sizes typically result in higher throughput for all block operations, but also higher latency and more memory for KV-cache storage, which forces a reduction in memory to meet the constraints (such as reducing the number of KV heads). For each set of deployment constraints, we choose the batch size that produced the highest quality architectures. If the target scenario specifies a maximum batch size, such as the typical number of active users for a chat bot, the search can be capped accordingly.

While MIP problems are NP-complete, modern solvers can efficiently handle instances of our size. Using the open-source `python-mip` package [25], we obtain high-quality solutions within seconds. This efficiency enables us to explore multiple architecturally diverse solutions by adding a diversity constraint:

$$\sum_{i=1}^{L}\sum_{j=1}^{K_i} x_{i,j}y_{i,j} \leq \alpha \cdot L \quad \forall y \in Y,$$

where Y is the set of previous solutions and $\alpha \in [0, 1]$ controls the maximum allowed similarity. For example, with $\alpha = 0.8$, each new solution must differ from previous solutions in at least 20% of its layer choices. This constraint helps discover meaningfully different architectures.

A key feature of our approach is its ability to generate solutions precisely tailored for specific hardware platforms. For example, in platforms with limited memory intended for batch size 1, the algorithm strongly favors memory-saving techniques such as FFN pruning, while de-prioritizing KV-cache optimizations which have minimal impact at batch 1. The hardware specificity extends to architectural features of the inference devices - for example, on H100 GPUs, FP8 quantization offers $\sim$2× acceleration and can be used aggressively, while on A100 GPUs where FP8 is unavailable, different optimization strategies must be employed. Even the difference in inter-GPU bandwidth between H100 PCI-E and NVLink configurations influences the optimal architecture by affecting tensor parallel synchronization costs.

This flexibility enables a powerful "train once, adapt on demand" methodology that requires minimal human intervention. After building a block library and computing block scores once, we can efficiently generate different architectures optimized for various deployment scenarios without additional training or manual tuning. The user needs only to specify the available block configurations for each platform - the hardware-specific measurements of block variants naturally guide the optimization toward platform-appropriate solutions. This approach makes Puzzle particularly valuable for real-world deployment, where the same parent model might need to be optimized differently across diverse hardware configurations and deployment constraints.

5 Post-Puzzle Inter-Block Uptraining

As mentioned in Section 3, the compatibility between adjacent blocks is not accounted for during the BLD step. Individual blocks are not fed with the outputs of their predecessor block in the child model, but rather with the output of the previous corresponding block in the parent model. As a result, after BLD, each block may receive an input distribution different from the one on which it was trained, resulting in sub-optimal performance. To mitigate this, the last stage in our framework consists of a short end-to-end training of the student using global knowledge distillation (GKD). GKD, commonly referred to as Knowledge Distillation, is a technique where a "student" model is trained to replicate or improve the performance of a "teacher" model by learning from its outputs and/or intermediate representations. In the context of this paper, the parent model serves as the teacher and the child model as the student, as the child seeks to inherit the distribution learned by the parent. Earlier methods [20] rely on cross-entropy loss between the teacher and student logits. Later approaches use Kullback-Leibler Divergence (KL Div) loss combined with language modeling loss (i.e. cross-entropy loss on next token prediction), facilitating a faster, more stable, and better performing optimization process [38, 33].

The optimal loss composition depends on the models selected, the training data, and the downstream tasks. We conducted a comprehensive evaluation of various combinations of three types of losses. The **Language Modeling (LM) Loss** quantifies the divergence between the predicted token probabilities and the target token probabilities in a supervised manner, calculated using the cross-entropy formula:

$$\mathcal{L}_{\text{LM}} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i), \tag{1}$$

where y_i represents the ground truth label for the i-th token in the vocabulary, $\hat{y}_i$ is the predicted probability distribution, and N is the size of the vocabulary. The **Cosine Similarity Loss** ensures similarity between the hidden representations of the parent and child models by maximizing the cosine similarity between their respective hidden states. This loss was also used by [38]:

$$\mathcal{L}_{\text{cosine}} = \sum_{l=1}^{L} \left(1 - \frac{\mathbf{h}_c^l \cdot \mathbf{h}_p^l}{\|\mathbf{h}_c^l\| \|\mathbf{h}_p^l\|} \right), \tag{2}$$

where $\mathbf{h}_c^l$ and $\mathbf{h}_p^l$ represent the hidden state vectors from the l-th transformer layer of the child and parent models, respectively, with L denoting the total number of transformer layers. While we assume a one-to-one correspondence between teacher and student blocks, this is not a requirement of the framework. Lastly, the **Kullback-Leibler Divergence (KLD) Loss** aligns the predicted logits of the child model with those of the parent model to facilitate knowledge transfer. It is computed as:

$$\mathcal{L}_{\text{KLD}} = \sum_{i=1}^{N} p_i \log \left(\frac{p_i}{q_i} \right), \tag{3}$$

where p_i and q_i represent the probability distributions derived from the logits of the parent and child models, respectively.

The final loss is the sum of the selected components in each experiment. All experiments were run on Nemotron-51B, our Llama-3.1-70B-Instruct derivative right after BLD, for approximately 5 billion tokens, which provided sufficient data to observe the relative effects of each loss combination. Our final model was trained on $45B$ tokens using the highest-scoring loss combination:

$$\mathcal{L}_{\text{GKD}} = \mathcal{L}_{\text{cosine}} + \mathcal{L}_{\text{KLD}} . \tag{4}$$

As shown in Table 1, LM loss is unnecessary for optimal uptraining and even **harms** downstream tasks in most cases. This observation aligns with findings from [38], and might result from overfitting to the dataset used during uptraining. Removing the LM loss helps mitigate the inherent data distribution mismatch (open-weights, closed-data) induced by having access only to the parent model, but not to the data on which it was originally trained. We show the importance of the GKD training stage in Table 13.

Table 1: Ablation study for different combinations of LM loss, block (hidden activations) loss, and logits KLD loss. All models (Nemotron-51B, derived from Llama-3.1-70B-Instruct) were trained for $\sim 5B$ tokens. First row did not undergo uptraining. Adjacent rows with the same color differ only in the $\mathcal{L}_{\text{LM}}$ component. *During the KD process for this combination, the validation $\mathcal{L}_{\text{KLD}}$ consistently increased. †Trained for 45B tokens using $\mathcal{L}_{\text{GKD}}$ defined in Equation (4).

$\mathcal{L}_{\text{LM}}$ (1)	$\mathcal{L}_{\text{cosine}}$ (2)	$\mathcal{L}_{\text{KLD}}$ (3)	MMLU	MT-Bench	Average	Validation $\mathcal{L}_{\text{KLD}}$
✗	✗	✗	78.39	8.67	82.55	0.19
✓	✗	✗	78.55	7.71	77.83	0.31*
✓	✗	✓	79.26	8.85	83.88	0.14
✗	✗	✓	79.33	8.68	83.07	**0.10**
✓	✓	✗	79.04	7.80	78.52	0.30*
✗	✓	✗	79.40	8.74	83.40	0.16
✓	✓	✓	79.45	8.66	83.03	0.14
✗	✓	✓	**79.61**	**8.87**	**84.16**	0.11
Llama-3.1-70B-Instruct (parent)			81.66	8.93	85.48	0.00
Nemotron-51B-Instruct (child)†			80.20	8.99	85.10	0.08

6 Supporting Fast Inference for Variable-Block Architectures in TensorRT-LLM

TensorRT-LLM is a highly optimized LLM runtime designed to accelerate inference performance on NVIDIA GPUs. It provides industry-leading performance for both latency and throughput-oriented workloads. The runtime enables LLMs to run on GPUs while utilizing custom paged attention [28] kernels to efficiently manage KV caching across sequences in a batch. Furthermore, TensorRT-LLM supports FP8 quantization and various scheduling policies that allow LLM deployments to optimally utilize the underlying hardware.

To enable the use of Puzzle-generated architectures from the search space (see Section 2) a major underlying assumption of TensorRT-LLM had to be revised: that all attention layers contain the same number of key and value heads. We devised changes to the paged KV cache strategy that enabled variable GQA ratios within a model. TensorRT-LLM now supports running any architecture from our search space including variable blocks and linear replacements, using FP8 precision for weights, activations and KV cache. Our NAS framework is designed to generate models that run efficiently in real inference scenarios. Therefore full support in inference engines and awareness to runtime considerations when applying on NAS scheme contributes greatly to the usability of resulting models, such as Nemotron-51B.

7 Main Results

Using our Puzzle framework, we generated Nemotron-51B as a child derivative of the Llama-3.1-70B-Instruct model. Nemotron-51B achieves a significant improvement in inference efficiency while retaining nearly all the accuracy of its parent model, demonstrating the effectiveness of our approach.

Evaluating Model Performance: To evaluate Puzzle-derived child models like Nemotron-51B, two performance metrics are of primary interest: 1) **Accuracy Preservation**: This measures how much of the parent model's accuracy is retained by the child. A high retention percentage indicates that the Puzzle framework successfully produces high-performing children. 2) **Computational Efficiency**: This reflects the child model's ability to adhere to the constraints it was optimized for. In our case, the focus is on *throughput*, showing how much we improved the model's suitability for reducing inference cost.

Together, these metrics demonstrate the balance between maintaining model quality and achieving optimization for deployment-specific needs.

Accuracy comparison: Table 2 compares the accuracy of Nemotron-51B with its parent across several benchmarks. On average, Nemotron-51B retains 98.4% of its parent's accuracy, even exceeding it on certain benchmarks such as MT-Bench and GSM8K Chat. This underscores the robustness of the Puzzle framework in maintaining model quality despite **significant** architectural modifications.

Table 2: Accuracy comparison of Nemotron-51B with Llama-3.1-70B-Instruct across several benchmarks. Accuracy preserved is the ratio of child to parent accuracy. *Chat prompt as defined in Adler et al. [2]. **version by CodeParrot.

Benchmark	Llama-3.1-70B-Instruct	Nemotron-51B	Accuracy Preserved (%)
Winogrande [43]	85.08	84.53	99.35
ARC Challenge [11]	70.39	69.20	98.30
MMLU [19]	81.66	80.20	98.21
HellaSwag [52]	86.44	85.58	99.01
GSM8K [12]	92.04	91.43	99.34
TruthfulQA [31]	59.86	58.63	97.94
XLSum English [17]	33.86	31.61	93.36
MMLU Chat*	81.76	80.58	98.55
GSM8K Chat*	81.58	81.88	100.37
Instruct HumanEval (n=20) [10]**	75.85	73.84	97.35
MT-Bench [53]	8.93	8.99	100.67

Human evaluation: We complement the above accuracy performance benchmarks with a human evaluation comparing the parent and child models. A blind test comparison of the models was conducted on a general purpose test set that includes tasks such as reasoning, longform text generation, knowledge and more. Results show that the difference between the models is *not* statistically significant (Figure 4), strengthening the claim that the accuracy degradation is minimal. For more details about the evaluation see appendix A.1.

Throughput comparison: Table 3 specifies the throughput performance of Nemotron-51B against its parent model across diverse input-output sequence lengths. Nemotron-51B achieves speedups up to 2.17x, enabling larger workloads per GPU and making it highly efficient for deployment.

Accuracy vs. throughput frontier: As the field of generative AI evolves, the tradeoff between accuracy and efficiency becomes a key factor in model selection, directly impacting deployment costs. Nemotron-51B is designed to achieve this balance, pushing beyond the current efficient frontier. Throughput is inversely proportional to cost, making Nemotron-51B the model with the best accuracy per dollar tradeoff, as shown in Figure 5. To account for both knowledge and conversational capabilities, accuracy is measured as a weighted combination of MMLU and MT-Bench scores: (MT-Bench $\times$ 10 + MMLU) / 2.

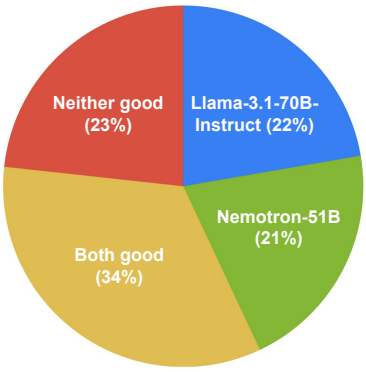


Figure 4: Preference of human annotators in a blind test comparison. Results indicate comparable performance between Llama-3.1-70B-Instruct and Nemotron-51B.

Table 3: Throughput comparison of Nemotron-51B and Llama-3.1-70B-Instruct across various scenarios. Throughput is measured in tokens per second per GPU (NVIDIA H100). TP# indicates the number of GPUs used in tensor parallelism. Note: Results were obtained on NVIDIA H100 SXM GPUs with FP8 quantization for weights, activations and KV cache using TensorRT-LLM. Optimal tensor parallelism was used for each model. Input/output sequence lengths indicate the prefill (input) and decode (output) operations performed by the LLM.

Scenario	Input/Output	Nemotron-51B (TP#)	Llama-3.1-70B-Instruct (TP#)	Speedup
Chatbot	128/128	5478 (TP1)	2645 (TP1)	2.07
Text Generation	128/1024	6472 (TP1)	2975 (TP4) / 1274 (TP1)	**2.17 / 5.08**
Long Text Generation	128/2048	4910 (TP2)	2786 (TP4)	1.76
Inference-time compute	128/4096	3855 (TP2)	1828 (TP4)	2.11
Summarization/RAG	2048/128	653 (TP1)	339 (TP4) / 301 (TP1)	1.92 / 2.17
Stress Test	2048/2048	2622 (TP2)	1336 (TP4)	1.96

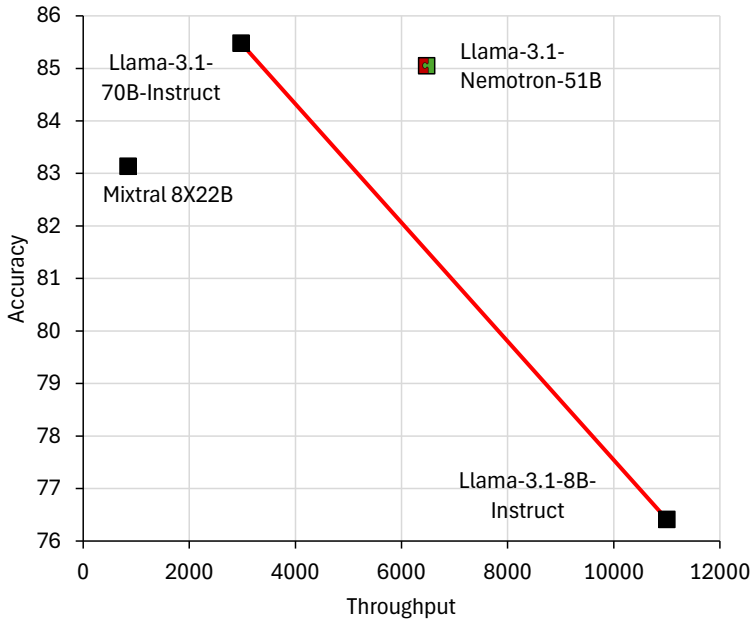


Figure 5: Accuracy vs. Throughput performance of Nemotron-51B compared to state-of-the-art models. Throughput is measured on NVIDIA H100 GPUs with the optimal TP setting per model, all running in FP8 on a "text generation" scenario (see Table 3). The red line represents the efficient frontier, highlighting models with the best accuracy-to-throughput tradeoff. Accuracy=(MT-Bench $\times 10$ + MMLU) / 2

The Nemotron-51B architecture achieves substantial computational savings through strategic reduction of computation across many layers, as shown in Figure 6. Observing the figure it is evident that our framework discovered significant optimization opportunities in both early layers (0-15) and later layers (45-70), with computed savings shown in green. Different regions exhibit distinct optimization patterns: early layers show balanced reduction in both attention and FFN components, while later layers demonstrate more aggressive attention optimization. Notably, the framework identifies a central region (layers 16-42) that maintains full computational capacity, suggesting these middle layers are critical for preserving model performance. This automatically discovered structure demonstrates that large efficiency gains are achievable through careful targeting of computational reduction, while maintaining essential computation where it matters most.

Evaluating Long-Context Performance: To evaluate the long-context performance of Nemotron-51B, we employed a subset of the RULER benchmark suite [23]. Llama-3.1-70B-Instruct, with a maximum context length of 128K tokens, serves as the performance upper bound. Notably, the child model was only exposed to sequences up to 8K tokens during training. Despite this limitation, the child model exhibited remarkable and somewhat surprising performance, retaining 96.23% of its parent's accuracy at 16K tokens across the RULER benchmark. This outcome highlights the robustness of Puzzle's framework in enabling generalization beyond the direct training horizon. Specifically, it is noteworthy that a child model trained solely on sequences up to 8K tokens could maintain such high accuracy at double the context length. We hypothesize that a short fine-tuning phase on longer contexts could further extend the model's effective context length.

At 32K tokens, the child model maintained a significant portion of the teacher's capabilities before performance degraded at 64K and beyond. This result aligns with expectations, given the child model's exposure constraints during training. These findings highlight the potential for efficient training strategies to extend the effective context length of language models beyond their direct training horizon.

Table 4 compares the performance of the parent and child models on benchmarks up to 16K tokens. Full performance details for extended context lengths are provided in Appendix B.

Table 4: Performance comparison of Llama-3.1-70B-Instruct (parent) and Nemotron-51B (child) on the RULER benchmark for context lengths up to 16K tokens. Accuracy preserved is the ratio of the Nemotron average to the Llama average, expressed as a percentage.

Context Length	Parent Average Score	Child Average Score	Accuracy Preserved (%)
1024	99.81	99.78	**99.90**
2048	97.97	97.32	**99.34**
4096	93.38	92.12	**98.65**
8192	92.68	90.63	**97.79**
16384	91.08	87.65	**96.23**

Finally, we applied the Puzzle framework to produce a child derivative of Llama-3.1-8B-Instruct, optimized for throughput specifically on an NVIDIA RTX 4090 GPU. This model breaks the efficient frontier for its throughput range, demonstrating Puzzle's ability to deliver highly efficient architectures also on consumer-grade hardware, while preserving the balance between accuracy and performance. Table 5 highlights this model's superior tradeoff in accuracy and efficiency.

Table 5: Accuracy and throughput of our high-throughput child derivative of Llama-3.1-8B-Instruct, which achieves equivalent throughput to Llama-3.2-3B-Instruct and far better accuracy. Throughput is estimated via the sum of measured block runtimes on a single NVIDIA RTX 4090 GPU, measured with an input-output sequence length of 1024 tokens each, the scenario for which this model was optimized. Accuracy = (MT-Bench $\times 10$ + MMLU) / 2.

Model	Throughput*	Accuracy
Ours (child)	5856	73.98
Llama-3.2-3B-Instruct	5737	70.36
Llama-3.1-8B-Instruct (parent)	3385	76.40

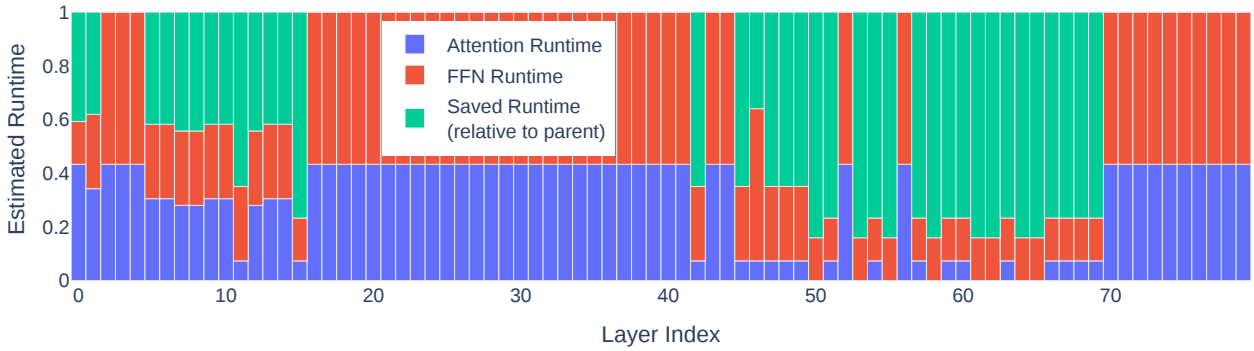


Figure 6: The estimated runtime of the attention and FFN subblocks of Nemotron-51B, relative to the original subblocks of Llama-3.1-70B-Instruct.

8 In-Depth Analysis and Ablation Studies

To better understand the key components and design choices of the Puzzle framework, we conduct a series of detailed analyses and ablation studies. We evaluate the importance of global knowledge distillation, investigate the impact of training dataset size and composition, and analyze how our MIP solver adaptively chooses architectures under varying constraints. These studies not only validate our design decisions but also provide insights into the fundamental trade-offs in LLM architecture optimization and the relative importance of different architectural components.

8.1 Block Library Construction Ablation Studies

In Sections 8.1.1 to 8.1.5, we analyze the impact of critical decisions in block library construction. These studies examine the fundamental trade-offs between computational cost, dataset characteristics, and model performance. Our key findings are:

- **Coupled vs. Decoupled BLD:** Decoupled BLD reduces training cost by transforming the search space from multiplicative to additive. Combining decoupled BLD for subspace narrowing with coupled BLD for refinement improves accuracy while maintaining computational costs (see Section 8.1.1).
- **Dataset Composition:** Models trained on the diverse Distillation Mix outperformed those trained on the limited-domain Project Gutenberg, but Gutenberg-trained models still retained 93% of performance, showcasing Puzzle's robustness (see Section 8.1.2).
- **Training Dataset Size:** BLD achieves strong performance even with smaller token budgets, with diminishing returns beyond 0.5B tokens (see Section 8.1.3).
- **Block Scoring Metrics:** KL divergence scoring outperformed LM loss and task-specific downstream scoring, demonstrating better balance between generality and accuracy, although task-specific scoring provides an advantage for customized target tasks (see Section 8.1.4).
- **Reduced Search Space:** Constraining the search space to no-op alternatives simplifies optimization and eliminates BLD but results in lower accuracy (75.4 vs. 78.39 MMLU), highlighting the value of diverse block replacements for optimal performance (see Section 8.1.5).

8.1.1 Combining Coupled BLD and Decoupled BLD

We investigate the effects of coupling in BLD (see Section 3.1) on Puzzle derivatives of Llama-3.1-8B-Instruct, which has 32 layers. For each layer, the search space we consider contains $|\mathcal{A}_i| = 6$ variants of the attention subblock and $|\mathcal{F}_i| = 12$ variants of the FFN subblock. With coupled BLD, this would amount to training $6 \cdot 12 \cdot 32 = 2304$ blocks. Decoupled BLD reduces the training requirements to only $(4 + 10) \cdot 32 = 448$ subblocks, which is considerably more resource-efficient. Note that besides moving from multiplicative composition to additive composition, with decoupled BLD we also do not need to train the no-op and parent variants in $\mathcal{A}_i$ and $\mathcal{F}_i$.

We propose a two-stage technique to reap the benefits of coupled BLD while still keeping the computational cost reasonable. First, we run the full Puzzle framework with decoupled BLD. Then, we analyze the architectures produced by the search algorithm, and identify the most prominent choices of subblock variants. We shrink the search space to include only these choices, then run the full Puzzle framework with coupled BLD on the reduced search space. In our

case, we were able to shrink $|\mathcal{A}_i|$ from 6 to 4, and $|\mathcal{F}_i|$ from 12 to 3, resulting in a total number of $4 \cdot 3 \cdot 32 = 384$ blocks to train in coupled BLD, which is similar to the number we trained in decoupled BLD for the larger search space. This approach produced a higher-accuracy architecture. Note, however, that the one created with decoupled BLD was already significantly above the efficient frontier. See results in Table 6. Both models underwent short GKD uptraining.

Table 6: The effect of coupled BLD vs decoupled BLD on high-throughput child derivatives of Llama-3.1-8B-Instruct. We found a relevant subspace of the search space using a decoupled BLD Puzzle, then trained coupled BLD on this subspace and ran a separate Puzzle, leading to additional improvement. Throughput is estimated via the sum of measured block runtimes on a single NVIDIA RTX 4090 GPU. Accuracy = (MT-Bench $\times 10$ + MMLU) / 2.

Model	Throughput*	Accuracy
Puzzle with Coupled BLD	5856	73.98
Puzzle with Decouple BLD	5834	72.92
Llama-3.2-3B-Instruct	5737	70.36

8.1.2 Impact of Dataset Composition on Puzzle-Derived Models

We evaluate the robustness of the Puzzle framework to training data composition by comparing performances up through the BLD stage (prior to GKD). For this analysis, we contrast two datasets: our domain-diverse Distillation Mix (described in Section 3) and the English subset of Project Gutenberg [41]. The latter, a dataset predominantly comprising literary works, which lacks diverse coverage of technical, conversational and STEM-specific content, making it an interesting test case for framework robustness.

As shown in Table 7, models derived using Project Gutenberg data (for training, pruning and block scoring) demonstrate strong performance despite the dataset's limitations. On general benchmarks like MT-Bench and MMLU, the Gutenberg-trained model achieves 92.7% and 95.5% of the performance obtained with Distillation Mix, respectively. Even on STEM categories within MMLU, where the training data's limitations are most relevant, the model maintains 91.7% of the performance (64.5 vs 70.35).

These results demonstrate that the Puzzle framework can effectively transfer knowledge from the parent model even when the training data provides limited coverage of specific domains. This robustness is particularly noteworthy given that no GKD uptraining was performed, suggesting that our BLD approach effectively preserves model capabilities across domains regardless of the training data composition.

Table 7: Benchmark results on Llama-3.1-70B-Instruct derivatives obtained from Puzzle without uptraining applied with different datasets.

Model	MT-Bench	MMLU	MMLU-STEM
Gutenberg-Trained	7.98	74.84	64.5
DistillationMix-Trained	8.61	78.39	70.35

8.1.3 Impact of Blockwise Local Distillation Training Dataset Size

To evaluate the efficiency of BLD under varying training dataset sizes, we conducted experiments using different token budgets. Specifically, we trained the same set of block variants using 0.25B, 0.5B, and 1.0B tokens from the *Distillation Mix* dataset (see Section 3). After completing the BLD stage for each token budget, we used the MIP optimization stage to generate optimized architectures under similar constraints to those used to produce Nemotron-51B, and evaluated their performance on downstream tasks.

In Table 8 we present the performance of models trained with different BLD token budgets. Notably, while all models achieved comparable MMLU scores, the MT-Bench results indicate a more pronounced performance improvement as the token budget increases, particularly in multi-turn conversational tasks. However, the improvements diminish as the token budget grows (e.g., the boost from 0.25B to 0.5B tokens is larger than that from 0.5B to 1.0B), suggesting that BLD facilitates rapid recovery of the parent model's performance even with moderate training budgets. These findings imply that longer BLD training can yield better block libraries but with diminishing returns at larger scales.

Table 8: Performance comparison of Puzzle-optimized architectures trained with varying BLD token budgets. Metrics include MT-Bench and MMLU scores.

BLD Token Budget	MT-Bench	MMLU
1.0B Tokens	8.98	78.54
0.5B Tokens	8.86	78.44
0.25B Tokens	8.51	78.27

8.1.4 Impact of Different Block Scoring Metrics

In Section 4.2, we presented three possible metrics for replace-1-block scores: downstream accuracy, LM loss and KL divergence. We investigate the effects of using different replace-1-block scores when applying the Puzzle framework to Llama-3.1-8B-Instruct with a large search space ($|\mathcal{A}_i| = 6, |\mathcal{F}_i| = 12$). We compare the use of the model's loss on a validation set (LM loss in our case), a common choice in earlier decomposed NAS methods used in CV, with our proposed method based on KL divergence. LM loss aims to capture the general quality degradation induced by changing a specific parent block, while KL divergence aims to capture the distance from the parent model induced by this change. As illustrated in Figure 7, KL divergence scoring results in better Puzzle architecture choices than scoring with LM loss. All models were constructed using decoupled BLD and underwent short GKD uptraining.

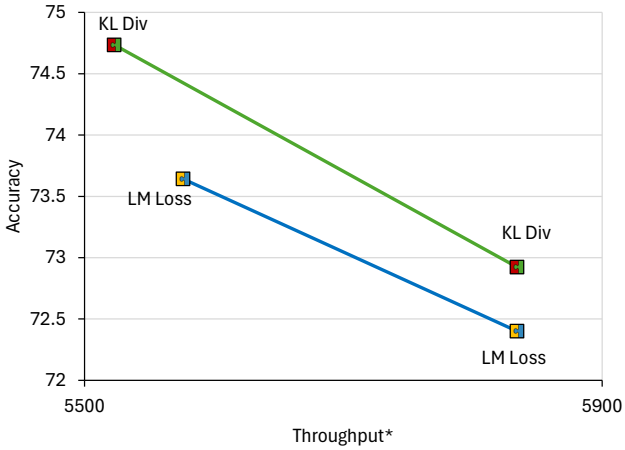


Figure 7: Accuracy vs. Throughput performance of Llama-3.1-8B-Instruct child derivatives, some constructed using LM loss as replace-1-block scores, and some constructed using KL divergence as replace-1-block scores. LM loss aims to capture the general quality degradation induced by changing a specific parent block, while KL divergence aims to capture the distance from the parent model induced by this change. KL divergence results in better architecture choices. Accuracy = (MT-Bench $\times 10$ + MMLU) / 2. Throughput is estimated via the sum of measured block runtimes on a single NVIDIA RTX 4090 GPU.

Next, we explore the use of downstream accuracy as replace-1-block scores to customize the algorithm's block selection for specific target tasks. Our hypothesis is that, within the same budget, different architectures may excel at different tasks because distinct capabilities—such as reasoning, world knowledge, or conversational abilities—are likely concentrated in different parts of the model. Due to the computational expense of calculating downstream accuracy, we use the reduced search space from Section 8.1.1 to keep the experiment feasible. To evaluate downstream accuracy, we use the MMLU benchmark, splitting its 57 tasks into two nearly equal-sized sets stratified by the MMLU categories {STEM, Social Sciences, Humanities, Other}. These subsets are referred to as *Half-MMLU*, with one serving as a "train" set for block quality scoring and the other as a "test" set for evaluation. Table 9 shows that, even with the same library of trained blocks, the block selection process can be customized to construct architectures optimized for specific target tasks. We note that this customization is not without cost: the architecture constructed using Half-MMLU scores achieves the best accuracy on the target task, but its MT-Bench accuracy of 7.57 is lower than the 8.06 MT-Bench accuracy of the architecture constructed according to the more general KL divergence scores. Both models were constructed using decoupled BLD and underwent short GKD uptraining.

Table 9: The effect of task-oriented block scoring on high-throughput child derivatives of Llama-3.1-8B-Instruct. We split the tasks in MMLU into two equal-sized sets and use one of them for block quality scoring and the other for evaluation, showing that even with the same library of trained blocks, block selection can be customized to build architectures that fit a desired target task. Throughput is estimated via the sum of measured block runtimes on a single NVIDIA RTX 4090 GPU.

Model	Throughput*	Half-MMLU Accuracy (Test Set)
Puzzle: scored with Half-MMLU accuracy (train set)	5818	66.24
Puzzle: scored with KL divergence	5834	64.94
Llama-3.2-3B-Instruct	5737	60.06

8.1.5 Effects of Limited Search Space Diversity

To explore the impact of reducing the search space complexity on the Puzzle framework, we constrained alternative child blocks to only allow replacing parent model subblocks with no-op operations. This eliminates the need for BLD as no additional block variants that require training are considered, further reducing the computational costs.

The resulting architecture, optimized using the same MIP approach but limited to no-ops, was evaluated against pre-uptraining Nemotron-51B, which was derived using the same Puzzle pipeline but with a more diverse block variants. As shown in Table 10, the no-op-only model retained high throughput but exhibited a noticeable drop in MMLU accuracy compared to Nemotron-51B (75.4 vs. 78.39).

These results illustrate the flexibility of the Puzzle framework in balancing resource constraints and model performance. Although limiting the search space simplifies the optimization process and reduces training costs even further, it sacrifices the fine-grained architectural customization enabled by a broader range of block alternatives. This demonstrates that a more diverse search space leads to architectures that achieve better accuracy.

Table 10: Comparison of pre-uptraining Nemotron-51B (derived using the full search space) and a no-op-only variant.

Model	MMLU	Throughput (tokens/sec)
Puzzle (No-op only)	75.40	5604.18
Puzzle (Full search space)	78.39	5500.25

8.2 Search Algorithm Ablation Studies

In Sections 8.2.1 to 8.2.3, we analyze the MIP optimization process and alternative approaches within the Puzzle framework. These studies focus on how throughput constraints and scoring strategies influence architecture selection and model performance. Our key findings are:

- **MIP Optimization and Throughput Constraints:** The MIP solver adapts architectures to meet various throughput targets, revealing nuanced trade-offs between global and local efficiency. Notably, FFN components are preserved even under strict constraints, highlighting their critical role in maintaining model accuracy (see Section 8.2.1).
- **Greedy Algorithm Alternative:** A budget-constrained greedy algorithm, while simpler, resulted in significantly lower accuracy (70.74% MMLU) compared to MIP-based optimization (78.39%). This underscores the importance of global optimization for achieving superior accuracy-efficiency trade-offs (see Section 8.2.2).
- **Data-free Scoring:** Maximizing parameter count as a heuristic produced architectures with sharp performance declines (23.12% MMLU), emphasizing the necessity of data-driven scoring mechanisms for effective architecture optimization (see Section 8.2.3).

8.2.1 MIP Solution Analysis Under Varying Throughput Constraints

Figure 8 provides an interesting window into how our MIP solver adapts model architecture to different throughput requirements. Each row in the heatmaps represents a distinct architecture optimized for a specific throughput target, with darker colors indicating higher computational cost relative to the parent model. The architecture of Nemotron-51B, corresponding to a throughput target of 5500 tokens per second, is marked in green. Several intriguing patterns are evident:

- **Counter-intuitive local optimizations:** While stricter throughput constraints generally lead to faster blocks, we observe surprising inversions of this trend. For example, in layers 1-4, the MIP sometimes chooses computationally heavier blocks for higher throughput targets. This counter-intuitive choice suggests that local slowdowns can enable better global optimization, with other layers compensating to meet the overall throughput constraint.
- **Asymmetric treatment of components:** The MIP treats attention and FFN components quite differently. While attention mechanisms are completely eliminated in some layers even under lenient throughput constraints, FFN components are never entirely skipped. This suggests that FFN layers might play a more fundamental role in maintaining model capabilities, while attention mechanisms offer more flexibility for optimization.
- **Architectural phases:** The heatmap reveals distinct "phases" across layer depths. Shallow layers (0-15) show high variability in both attention and FFN, middle layers (16-42) maintain more consistent computation, and deeper layers (43-80) show different patterns for attention versus FFN optimization. This suggests different layers serve distinct roles in the network's information processing.
- **Throughput-dependent transitions:** The solution patterns show clear transitions as throughput requirements increase, but these transitions are not uniform across layers. Some layers maintain consistent computation across different throughput targets while others show sharp transitions, indicating varying sensitivity to throughput constraints.

These patterns demonstrate the sophisticated optimization strategies discovered by the MIP solver, revealing that optimal architectures often require nuanced tradeoffs between local and global efficiency. The preservation of FFN computation even under strict constraints provides empirical evidence for the relative importance of different architectural components in such transformer models.

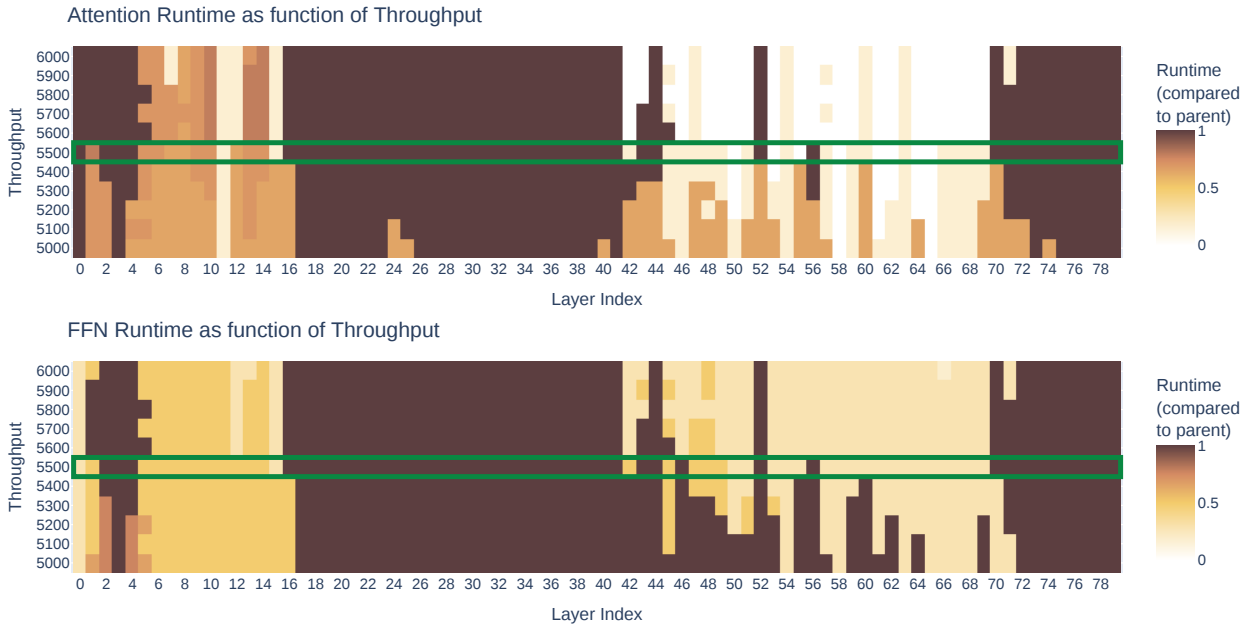


Figure 8: Heatmaps showing how attention and FFN runtime patterns vary with throughput constraints across model layers. Dark colors indicate higher computational cost relative to the parent model. Each row represents an architecture optimized for a specific throughput target, with Nemotron-51B's configuration (5500 tokens/sec) marked in green.

8.2.2 Greedy Search Algorithm Alternative

To further evaluate the impact of the MIP optimization approach, we implemented a budget-constrained greedy search algorithm as an alternative. This algorithm prioritizes simplicity by using a heuristic-based layer-wise selection process, contrasting with the global optimization provided by MIP.

The greedy algorithm operates as follows:

- At initialization, the runtime and memory budgets which are derived from the required constraints, are split equally across layers.
- Layer scoring: Each layer is assigned a score based on a heuristic metric designed to estimate how easily it can be replaced with minimal performance degradation. In our implementation, this metric was the mean replace-1-block KL divergence score across all block variants for the layer. Layers with lower average scores were deemed easier to optimize.
- Sequential replacement: Layers are processed in ascending order of their scores. For each layer, the algorithm selects the block variant with the lowest replace-1-block KL divergence score that satisfies the layer's runtime and memory budget.
- Constraint adjustment: After selecting a block for the current layer, the remaining runtime and memory savings are added to the next layer's budget. This allows for a more dynamic algorithm that allocates more resources for layers that are harder to replace.

Table 11 compares the performance of the greedy algorithm with the MIP-derived pre-uptraining Nemotron-51B. The results show that the greedy algorithm leads to a significant drop in accuracy, highlighting the importance of global optimization. Specifically, the model derived using the greedy algorithm achieves a substantially lower MMLU accuracy of 70.74%, compared to 78.39% for the MIP-derived model, despite both architectures meeting the same throughput constraint.

Table 11: Comparison of the budget-constrained greedy algorithm and MIP as search algorithms for Puzzle. Results are shown for pre-uptraining Nemotron-51B under identical throughput constraints.

Optimization Method	MMLU	Throughput (tokens/sec)
Greedy Algorithm	70.74	5500.30
MIP	78.39	5500.25

These findings underscore the critical role of global optimization in the Puzzle framework. By considering jointly optimizing block selection, MIP achieves a significantly better balance between efficiency and accuracy, making it indispensable for extracting the full potential of the Puzzle framework.

8.2.3 Importance of Data-Driven Quality Estimation in Architecture Scoring

To further understand the importance of quality estimation in search space optimization, we conducted an experiment using a simple heuristic: scoring blocks based on their parameter count. While not a data-driven approach, parameter count serves as a straightforward metric often associated with improved performance in LLMs. This experiment aimed at assessing whether such a basic metric could provide sufficient guidance in the search space, shedding light on the role of more nuanced scoring mechanisms.

Under this method, the search algorithm is simplified to selecting the block variant with the largest number of parameters that satisfied throughput and memory constraints. This resulted in an architecture composed uniformly of high-parameter blocks across all layers, without considering layer-specific block quality and representational needs.

As shown in Table 12, maximizing parameter count leads to a sharp drop in MMLU accuracy compared to the architecture derived using the full Puzzle framework with MIP optimization. Despite having similar throughput, the simplistic parameter-maximization approach fails to achieve competitive results, underscoring the necessity of quality-aware block scoring, and different layers require different computational budgets for optimal performance.

Table 12: Comparison of maximizing parameter count with Puzzle's MIP-based optimization as search algorithms. Results are shown for pre-uptraining Nemotron-51B under identical throughput constraints.

Optimization Method	MMLU	Throughput (tokens/sec)
Maximizing Parameters	23.12	5727.08
pre-uptraining Nemotron-51B	78.39	5500.25

8.3 Global Knowledge Distillation Uptraining Importance

We assess the significance of the final GKD uptraining phase, in enhancing the accuracy of child models derived from Llama-3.1-70B-Instruct and Llama-3.1-8B-Instruct. The results presented in Table 13 demonstrate that this stage contributes to improvements in both the MMLU and MT-Bench benchmark scores.

Table 13: Impact of global knowledge distillation uptraining on MMLU and MT-Bench benchmark scores for child models derived from Llama-3.1-70B-Instruct and Llama-3.1-8B-Instruct.

Model Name	GKD Uptraining	MMLU	MT-Bench	Average
Llama-3.1-70B-Instruct (parent)	-	81.66	8.93	85.48
Nemotron-51B-Instruct (child)	✗	78.39	8.67	82.55
	✓	80.20	8.99	85.10
Llama-3.1-8B-Instruct (parent)	-	69.40	8.34	76.40
Child derivative of Llama-3.1-8B-Instruct (child)	✗	65.25	7.29	69.06
	✓	65.46	8.25	73.98

9 Conclusions and Future Directions

In this work we present Puzzle, a framework that modifies LLMs from their over-parameterized training configurations to optimized, inference-efficient architectures tailored for specific hardware. Our results challenge the conventional wisdom that model architectures should remain uniform across training and inference, demonstrating that post-training architectural optimization can significantly improve efficiency without compromising performance. Using Puzzle, we developed Nemotron-51B from the Llama-3.1-70B-Instruct model, achieving unmatched throughput and memory efficiency on a single NVIDIA H100 GPU. Nemotron-51B maintains over 98% of the parent model's accuracy while delivering a 2.2x throughput improvement in optimized settings and up to 5x improvement on a single GPU, demonstrating the effectiveness of optimizing for deployment environments rather than focusing on model size alone.

The Puzzle framework achieves these improvements with remarkable efficiency in training resources. Requiring fewer than 50B tokens—compared to the trillions needed to train models from scratch—Puzzle produces high-performing models at a fraction of the usual cost. This extreme search and training efficiency still results in drastic inference performance improvements of Puzzle optimized models, thus enabling widespread and efficient deployment of LLMs across data centers and edge devices.

The success of Puzzle opens several promising directions for future research. Our introduction of decoupled BLD, which is significantly more efficient than coupled BLD, makes it feasible to evaluate a much larger set of potential blocks within a single optimization run. This efficiency enables exploration of novel operations as alternative Puzzle blocks, such as variable window attention mechanisms [9], *state-space models* [15, 14], or other architectural innovations. The framework could also be extended to optimize models for specific capabilities, such as Chain-of-Thought reasoning or multimodal tasks, including vision-language models [32, 54] and retrieval-augmented generation [29].

Future work might also explore more sophisticated search algorithms beyond MIP, incorporating reinforcement learning or evolutionary approaches to discover even more efficient architectures. The relationship between architectural patterns and specific model capabilities deserves deeper investigation, as understanding this connection could lead to more targeted optimization strategies. Additionally, studying the robustness of Puzzle-optimized models to distribution shifts and their ability to adapt to new tasks through fine-tuning would be valuable, particularly in understanding how architectural modifications affect model reliability and exploring methods for dynamic architecture adaptation.

Puzzle represents a significant step toward making advanced language models more practical and accessible. The framework's ability to customize state-of-the-art models based on hardware constraints, while maintaining high performance, points the way toward a future where powerful AI systems can be affordably deployed across a wide range of environments and use cases, advancing the field of generative AI in a cost-effective and application-oriented direction.

Acknowledgments

We thank Saurav Muralidharan, Sharath Turuvekere Sreenivas, and mainly Pavlo Molchanov for fruitful discussions.

References

[1] The claude 3 model family: Opus, sonnet, haiku. URL https://api.semanticscholar.org/CorpusID:268232499.

[2] Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan M. Cohen, Sirshak Das, Ayush Dattagupta, Olivier Delalleau, Leon Derczynski, Yi Dong, Daniel Egert, Ellie Evans, Aleksander Ficek, Denys Fridman, Shaona Ghosh, Boris Ginsburg, Igor Gitman, Tomasz Grzegorzek, Robert Hero, Jining Huang, Vibhu Jawa, Joseph Jennings, Aastha Jhunjhunwala, John Kamalu, Sadaf Khan, Oleksii Kuchaiev, Patrick LeGresley, Hui Li, Jiwei Liu, Zihan Liu, Eileen Long, Ameya Sunil Mahabaleshwarkar, Somshubra Majumdar, James Maki, Miguel Martinez, Maer Rodrigues de Melo, Ivan Moshkov, Deepak Narayanan, Sean Narenthiran, Jesus Navarro, Phong Nguyen, Osvald Nitski, Vahid Noroozi, Guruprasad Nutheti, Christopher Parisien, Jupinder Parmar, Mostofa Patwary, Krzysztof Pawelec, Wei Ping, Shrimai Prabhumoye, Rajarshi Roy, Trisha Saar, Vasanth Rao Naik Sabavat, Sanjeev Satheesh, Jane Polak Scowcroft, Jason Sewall, Pavel Shamis, Gerald Shen, Mohammad Shoeybi, Dave Sizer, Misha Smelyanskiy, Felipe Soares, Makesh Narsimhan Sreedhar, Dan Su, Sandeep Subramanian, Shengyang Sun, Shubham Toshniwal, Hao Wang, Zhilin Wang, Jiaxuan You, Jiaqi Zeng, Jimmy Zhang, Jing Zhang, Vivienne Zhang, Yian Zhang, and Chen Zhu. Nemotron-4 340b technical report. *CoRR*, abs/2406.11704, 2024. doi: 10.48550/ARXIV.2406.11704. URL https://doi.org/10.48550/arXiv.2406.11704.

[3] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568.

[4] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.298. URL https://doi.org/10.18653/v1/2023.emnlp-main.298.

[5] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR, 2019. URL http://proceedings.mlr.press/v97/allen-zhu19a.html.

[6] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/ARXIV.2312.11805. URL https://doi.org/10.48550/arXiv.2312.11805.

[7] Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari Do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=vXxardq6db.

[8] Mikhail Belkin, Daniel J. Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116:15849 – 15854, 2018. URL https://api.semanticscholar.org/CorpusID:198496504.

[9] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov,

Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

[11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018. URL http://arxiv.org/abs/1803.05457.

[12] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

[14] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

[15] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

[16] Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=CNL2bku4ra.

[17] Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Samin Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. Xl-sum: Large-scale multilingual abstractive summarization for 44 languages. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 4693–4703. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-ACL.413. URL https://doi.org/10.18653/v1/2021.findings-acl.413.

[18] Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is needed. *CoRR*, abs/2406.15786, 2024. doi: 10.48550/ARXIV.2406.15786. URL https://doi.org/10.48550/arXiv.2406.15786.

[19] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

[20] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[21] Hive-Digital-Technologies. https://huggingface.co/datasets/H-D-T/Buzz-V1.2.

[22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

[23] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: what's the real context size of your long-context language models? *CoRR*, abs/2404.06654, 2024. doi: 10.48550/ARXIV.2404.06654. URL https://doi.org/10.48550/arXiv.2404.06654.

[24] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[25] COIN-OR Foundation Inc. Python-mip: collection of python tools for the modeling and solution of mixed-integer linear programs, 2023. URL https://github.com/coin-or/python-mip.

[26] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

[27] Eldar Kurtic, Denis Kuznedelev, Elias Frantar, Michael Goin, and Dan Alistarh. Sparse fine-tuning for inference acceleration of large language models. *CoRR*, abs/2310.06927, 2023. doi: 10.48550/ARXIV.2310.06927. URL https://doi.org/10.48550/arXiv.2310.06927.

[28] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

[29] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401, 2020. URL https://api.semanticscholar.org/CorpusID:218869575.

[30] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1986–1995, 2019. URL https://api.semanticscholar.org/CorpusID:208513081.

[31] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.229. URL https://doi.org/10.18653/v1/2022.acl-long.229.

[32] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *ArXiv*, abs/2304.08485, 2023. URL https://api.semanticscholar.org/CorpusID:258179774.

[33] Chengqiang Lu, Jianwei Zhang, Yunfei Chu, Zhengyu Chen, Jingren Zhou, Fei Wu, Haiqing Chen, and Hongxia Yang. Knowledge distillation of transformer-based language models revisited. *arXiv preprint arXiv:2206.14366*, 2022.

[34] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/44956951349095f74492a5471128a7e0-Abstract-Conference.html.

[35] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. *CoRR*, abs/2403.03853, 2024. doi: 10.48550/ARXIV.2403.03853. URL https://doi.org/10.48550/arXiv.2403.03853.

[36] Pavlo Molchanov, Jimmy Hall, Hongxu Yin, Jan Kautz, Nicolò Fusi, and Arash Vahdat. LANA: latency aware network acceleration. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal

Hassner, editors, *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XII*, volume 13672 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2022. doi: 10.1007/978-3-031-19775-8_9. URL https://doi.org/10.1007/978-3-031-19775-8_9.

[37] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12209–12218, 2020. URL https://api.semanticscholar.org/CorpusID:229211047.

[38] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *CoRR*, abs/2407.14679, 2024. doi: 10.48550/ARXIV.2407.14679. URL https://doi.org/10.48550/arXiv.2407.14679.

[39] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

[40] Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. *CoRR*, abs/2406.17557, 2024. doi: 10.48550/ARXIV.2406.17557. URL https://doi.org/10.48550/arXiv.2406.17557.

[41] Project Gutenberg. https://www.gutenberg.org.

[42] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:743641.

[43] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

[44] Sunny Sanyal, Ravid Shwartz-Ziv, Alexandros G. Dimakis, and Sujay Sanghavi. Inheritune: Training smaller yet more attentive language models, 2024. URL https://arxiv.org/abs/2404.08634.

[45] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150.

[46] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Raghavi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Evan Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15725–15788. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.840. URL https://doi.org/10.18653/v1/2024.acl-long.840.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[48] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1513–1528. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.107. URL https://doi.org/10.18653/v1/2022.acl-long.107.

[49] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations, ICLR*

2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024. URL https://openreview.net/forum?id=09iOdaeOzp.

[50] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

[51] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

[52] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1472. URL https://doi.org/10.18653/v1/p19-1472.

[53] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

[54] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *CoRR*, abs/2408.11039, 2024. doi: 10.48550/ARXIV.2408.11039. URL https://doi.org/10.48550/arXiv.2408.11039.

[55] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016. URL https://api.semanticscholar.org/CorpusID:12713052.

A Appendices

A.1 Human Evaluation

A blind-test comparison between Nemotron-51B and Llama-3.1-70B-Instruct was conducted in the following manner:

- A set of 169 samples was sent.
- The evaluation was done by Nvidia's data factory team.
- Three annotators annotated each sample independently, resulting in a total of $169 \cdot 3 = 507$ annotations.
- Annotators saw [prompt, completion 1, completion 2] and had to choose between 4 options:
  - Completion 1 is better.
  - Completion 2 is better.
  - Both are good.
  - Neither is good.
- The order of the completions was randomized to avoid positional bias.

The test set was curated by the project's product team and Nvidia's data factory and included the following tasks and subtasks listed in brackets:

- Long form text generation (write a blog, write a story, other).
- Long inputs (Open book QA, Multi-hop questions, text to headline)
- Grounded text generation (table of contents to blog, paraphrasing).
- Multi-condition instructions (3-5 conditions, 6-10 conditions).

- Knowledge and trivia.
- Summarization (full document summary, summarize to bullet points, summarize paragraph to a sentence).
- Reasoning (temporal reasoning, cause and effect, navigation, general reasoning).
- Semantic extraction.

B RULER Benchmark Performance Tables

This section provides the complete performance tables for both the parent (Llama-3.1-70B-Instruct) and child (Nemotron-51B) models on a subset of the RULER benchmark across all context lengths evaluated.

Table 14: Full performance comparison of the parent (Llama-3.1-70B-Instruct) and child (Nemotron-51B) models on a subset of the RULER benchmark across all context lengths. Accuracy preserved is the ratio of the child score to the parent score, expressed as a percentage. The names of the benchmarks refer to their implementation and settings used in the official github repository. *Varies depending on context length

Context Length	qa_hotpotqa	qa_squad	common_words_extraction*	variable_tracking_1_4	variable_tracking_2_2	freq_words_extraction_2	freq_words_extraction_3.5	Average	Accuracy Preserved (%)
Parent									
1024	N/A	N/A	100.00	100.00	100.00	99.40	99.67	99.81	-
2048	N/A	88.40	100.00	100.00	100.00	99.53	99.87	97.97	-
4096	67.60	87.40	100.00	100.00	99.87	99.00	99.80	93.38	-
8192	67.80	83.80	99.96	100.00	99.40	97.87	99.93	92.68	-
16384	63.20	82.00	98.86	100.00	96.87	96.67	99.93	91.08	-
32768	61.60	77.20	93.48	100.00	97.93	95.53	100.00	89.39	-
65536	55.4	72.60	26.16	99.96	97.93	94.53	99.87	78.06	-
131072	33.65	49.04	2.37	56.85	36.33	78.61	85.71	48.94	-
Child									
1024	N/A	N/A	99.98	100.00	100.00	99.40	99.53	99.78	**99.9**
2048	N/A	86.20	99.86	99.96	99.67	98.40	99.80	97.32	**99.34**
4096	63.40	85.00	99.92	100.00	98.93	97.73	99.87	92.12	**98.65**
8192	58.20	80.80	99.34	100.00	99.60	96.67	99.80	90.63	**97.79**
16384	53.40	75.60	93.50	99.72	96.80	94.73	99.80	87.65	**96.23**
32768	45.60	70.60	51.92	98.28	93.67	90.27	99.47	78.54	**87.86**
65536	7.4	15.20	2.28	3.48	7.87	36.93	8.67	11.6	**14.86**
131072	3.80	3.20	0.10	0.00	0.00	2.07	0.00	1.31	**2.67**