`whittle`: A Library for Sub-Network Extraction from Large Language Models

Arjun Krishnakumar[*1] **Rhea Sanjay Sukthanker**[*1] **Hannan Javed Mahadik**[*3]
Gabriela Kadlecová[4] **Vladyslav Moroshan**[1] **Timur Carstensen**[3] **Frank Hutter**[2, 3, 1]
Aaron Klein[3]

[1]University of Freiburg
[2]Prior Labs
[3]ELLIS Institute Tübingen
[4]Distill Labs

Abstract We introduce `whittle`, an open-source library for extracting structured sub-networks from pretrained model checkpoints, providing a principled approach to initializing small language models and to model compression. Among its components, `whittle` exposes a sub-network extraction API alongside pretraining and knowledge-distillation pipelines, and is designed so that the extraction primitive can be driven by external search procedures over arbitrary user-defined sub-network search spaces. Using `whittle`, we identify sub-networks of Pythia-6.9B and Pythia-12B that match the validation perplexity of comparable Pythia baselines while requiring up to **5.16**× and **1.26**× fewer floating point operations at 10B and 100B tokens, respectively. We further demonstrate the generality of the extraction API on Qwen3-8B, where the same primitive supports three external search methods: random search, evolutionary search, and Bayesian optimization with the Tree-structured Parzen Estimator. We release the code of `whittle` at `https://github.com/whittle-org/whittle` and the code for the experiments at `https://github.com/whittle-org/whittle-paper`.

1 Introduction

Large Language Models (LLMs) achieved remarkable performance across a wide range of tasks. Their success is largely driven by scale: modern LLMs routinely exceed tens and hundreds of billions of parameters, unlocking outstanding generalization and emergent abilities. However, this scale comes at a cost. Training and deploying such massive models requires substantial computational resources, and inference often exceeds practical memory or latency budgets.

These challenges have motivated increasing interest in *Small Language Models (SLMs)* [Allal et al., 2025, Yang et al., 2025], which aim to preserve strong performance while remaining deployable in resource-constrained settings such as mobile or edge devices. Although pretraining SLMs is substantially cheaper than training LLMs, the costs are still considerable and often beyond the reach of most smaller research groups. For example, Allal et al. [2025] estimate that training SmolLM2 with 1.7B parameters required on the order of 10^{23} FLOPs – roughly $250,000 of GPU compute.

A common strategy to reduce pre-training cost is to extract *sub-networks* from open-weight LLMs and use them to warm-start SLM pretraining or fine-tune for a target task [Muralidharan et al., 2024, Meta AI, 2024, Liu et al., 2026]. The open-weight LLM can be used as a teacher model to perform distillation and further accelerate training. For example, Muralidharan et al. [2024] proposed pruning a teacher model to extract sub-networks which are then further refined through distillation. Similarly, the smaller variants of the Llama 3.2 family [Meta AI, 2024] were obtained using a combination of pruning and distillation. More recently the Ministral-3 [Liu et al., 2026] family of models were also obtained using iterative pruning and distillation.

 © 2026 the authors, released under CC BY 4.0

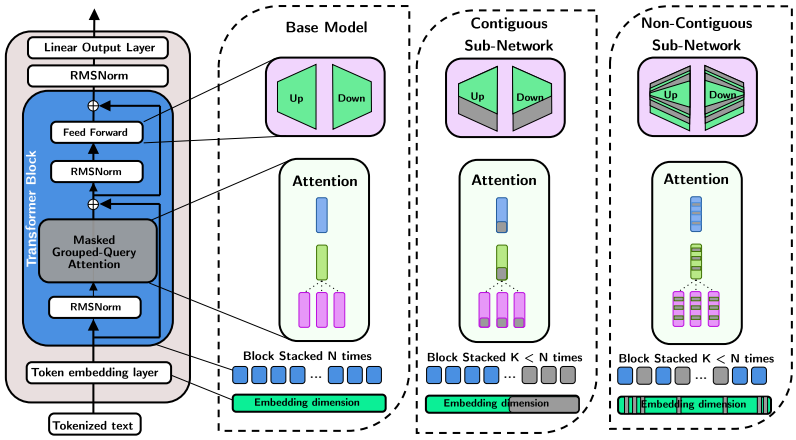


Figure 1: The two modes of sub-network extraction: contiguous and non-contiguous. The base model is composed of large matrices, several rows or columns of which are discarded to produce the smaller matrices. In contiguous mode, these matrices are simply sliced using a *range* of indices, whereas the non-contiguous mode samples random unique indices to select a subset of the larger matrix. The weights which are discarded are represented in gray.

While this line of work has great potential to accelerate SLM training, most existing efforts are closed-source. Given that progress in machine learning is heavily driven by open-source libraries and frameworks, the lack of open implementations for sub-network extraction makes it difficult to reproduce results, disentangle algorithmic advances from engineering choices, and fairly compare competing approaches.

In this paper, we present an open-source library, `whittle`, for sub-network extraction and pretraining. Our contributions are as follows:

- We introduce `whittle`, an open-source library for extracting structured sub-networks from pretrained model checkpoints, enabling model compression and a principled way to warm-start initialization of SLMs.
- We show that sub-networks extracted with `whittle` consistently outperform random initialization, reducing training compute by **5.16**× and **1.26**× and yielding average relative performance gains of up to **12.87**% and **9.8**% at 10B and 100B tokens, respectively.
- We present systematic ablations across search-space granularity (Figure 5), search methods (Figure 21), importance-scoring criteria (Figure 3), and distillation hyperparameters (Appendix, Figures 12–14), providing insights into when and why warm-starting from extracted sub-networks is effective.

Section 2 introduces our open-source library `whittle`, and Section 3 describes our search framework. We present our experiments and their results in Section 4, and conclude with a discussion of broader implications and future work in Section 5.

2 `whittle`: The Library

We now describe `whittle`, a lightweight library built on top of `litgpt`[1][AI, 2023], that enables sub-network extraction from base language models, Minitron-style importance sorting of the neurons of the model, and facilitates subsequent pretraining and distillation (see Figure 7 for an overview). Section 2.1 discusses the mechanism by which smaller networks can be obtained from base language models, and Section 2.2 discusses importance sorting. Then, in Section 2.3, we see

[1] `https://github.com/Lightning-AI/litgpt/`

how the library can be used for subsequent pretraining and distillation of these models. We provide overview of APIs for different functionalities of the library in Appendix G.

2.1 Sub-Network Extraction

Most language models share the same skeletal structure as shown in Figure 1 (left). The input tokens are first converted into an embedding of a fixed dimension d_{model} using a learned embedding layer *Emb*. These embeddings are then passed through a series of transformer blocks B_i, where the output of one block serves as the input to the next. Within each transformer block, masked attention $Attn_i$ is applied to the embeddings. Normalization (usually RMSNorm [Zhang and Sennrich, 2019]) may be applied before or after the attention mechanism (or both) depending on the model. Finally, these embeddings are passed through a feed-forward network F_i to produce the output of block B_i. This network usually contains a hidden inverted bottleneck dimension d_h and its output matches the embedding dimension d_{model} of the model.

By slicing the weight matrices in the base model we can extract a sub-network from it. Our library, `whittle`, does precisely this. We identify the four dimensions which can be shrunk to extract smaller networks from the base model: (1) The embedding layer *Emb* by reducing the embedding dimension d_{model}. (2) The number of transformer blocks N. (3) The intermediate dimension d_h^i of the feed forward network F_i in B_i. (4) The matrices associated with the attention mechanism $Attn_i$ in B_i.

2.1.1 Attention. Every attention module is described by three matrices — the query (Q), key (K), and value (V). These are computed from the embeddings matrix $X \in \mathbb{R}^{T \times d_{model}}$ as follows:

$$
\begin{aligned}
Q &= XW_Q, \quad & W_Q &\in \mathbb{R}^{d_{\text{model}} \times (hd_k)} \\
K &= XW_K, \quad & W_K &\in \mathbb{R}^{d_{\text{model}} \times (gd_k)} \\
V &= XW_V, \quad & W_V &\in \mathbb{R}^{d_{\text{model}} \times (gd_v)}
\end{aligned}
$$

where d_{model} is the embedding dimension of the model, T is the sequence length, h is the number of query heads, g is the number of key/value groups (which divides h), and d_k and d_v are the dimensions of the keys/queries and values.

Of these, the dimensions which can be modified to extract smaller attention mechanisms are: (1) The number of key/value groups, where entire groups of keys, values, and the associated queries can be discarded from the corresponding weight matrices. (2) The number of queries h can be reduced if the new number of queries h_{new} is also divisible by the number of groups g. (3) The dimensions of the keys/queries d_k and values d_v can be reduced. See Figure 8 in the Appendix for an overview of different attention mechanisms: *Multi-Head Attention (MHA)*[Vaswani et al., 2017], *Grouped-Query Attention (GQA)* [Ainslie et al., 2023], and *Multi-Query Attention (MQA)* [Shazeer, 2019]. Figure 9 illustrates how GQA can be converted into MHA and MQA.

2.1.2 Modes of sub-matrix extraction. Sub-networks are obtained from base models by extracting sub-matrices which may be either contiguous blocks in the larger matrix or index-selected rows and columns. `whittle` supports both these modes of sub-matrix extraction. Consider, for example, that we wish to reduce the embedding dimension from d_{model} in the base model to $\frac{d_{model}}{2}$ in the sub-network. This can be achieved by slicing the weights $W_E \in \mathbb{R}^{vocab_size \times d_{model}}$ of the embedding table *Emb* to $W_E^{sub} \in \mathbb{R}^{vocab_size \times \frac{d_{model}}{2}}$. A sub-matrix of this shape can also be obtained by sampling $\frac{d_{model}}{2}$ unique indices that fall in the range of $[0, d_{model} - 1]$ and using them to index W_E along its second dimension. This principle is applicable to all the matrices we consider, including attention weight matrices W_k^i, W_q^i and W_v^i, and the weights W_f^i of the feed-forward network F_i of a transformer block B_i. This *non-contiguous* mode of sub-matrix extraction makes it possible for the resulting sub-networks to sample its weights from the entire space of the larger base weights. Further,

`whittle` allows each transformer block to have its own shapes of weights, allowing researchers to experiment with a much larger diversity of model architectures.

2.2 Importance Sorting

Beyond raw sub-matrix extraction, `whittle` supports *Minitron-style* importance sorting [Muralidharan et al., 2024], which permutes the rows and columns of the base model's weight matrices so that the most important neurons and heads in each layer are placed at the lowest indices. Once the base model is sorted in this way, a contiguous sub-matrix extraction (slicing the first n indices along a dimension) automatically retains the components deemed most important, providing a strong initialization for the extracted sub-network without any additional training. See Appendix E.1 for more details about importance sorting.

2.3 Pretraining and Distillation

In this section we provide an overview of the pretraining and distillation functionality of `whittle`.

2.3.1 Pretraining. Being a wrapper around `litgpt`, `whittle` supports the pretraining pipeline that is already present in `litgpt` for language models. Given a configuration, the sub-network extraction functionality of `whittle` extracts the associated weights from the base model and instantiates a dense language model with the specified architecture. The resulting small language model (SLM) can then be used in the pretraining pipeline, where it is optimized using the standard token-level cross-entropy language modeling objective.

2.3.2 Knowledge Distillation. `whittle` includes built-in support for knowledge distillation, enabling extracted sub-networks to leverage pretrained teacher models during small language model training. Rather than training solely with hard labels, the library lets student models incorporate softened teacher outputs through a temperature-scaled softmax with temperature $T > 0$.

In `whittle`, distillation is configured through a composite training objective that combines a standard token-level cross-entropy loss $\mathcal{L}_{\text{CE}}$, computed between the hard target labels $\mathbf{y}$ and the student predictions $\mathbf{s}$, with a user-selected distillation loss $\mathcal{L}_{\text{D}}$, computed between the teacher and student output distributions p_t and p_s:

$$\mathbf{L} = \alpha\, \mathbf{L}_{\text{CE}}(\mathbf{s}, \mathbf{y}) + \beta\, \mathcal{L}_{\text{D}}(p_s, p_t), \tag{1}$$

where $\alpha, \beta \in [0, 1]$ balance hard-label supervision and distillation. The library supports multiple choices for $\mathcal{L}_{\text{D}}$ (Section D.2); in our experiments we use the forward Kullback–Leibler divergence between the temperature-softened teacher and student distributions. `whittle` also supports *top-k logit distillation* (Section D.2), where distillation is performed using only the highest-probability teacher logits, reducing memory and communication overhead while preserving informative supervision signals.

3 Search Framework

We now present our framework to search for sub-networks. We first describe our parameter binning approach to find sub-networks of a specific size (Section 3.1) and our search spaces (Section 3.2). Finally, we discuss the different search methods (Section 3.3) used in our experiments.

3.1 Parameter Bins

In practice, we usually aim to extract a sub-network with a specific parameter count. However, given the discrete nature of our search space, it is difficult to match a desired parameter count exactly. We use *parameter bins* to limit the search to a specific parameter range. Given a pretrained large language model (LLM) with parameters ($\boldsymbol{\theta} \in \mathbb{R}^S$), where $S = |\boldsymbol{\theta}|$ denotes the total number of

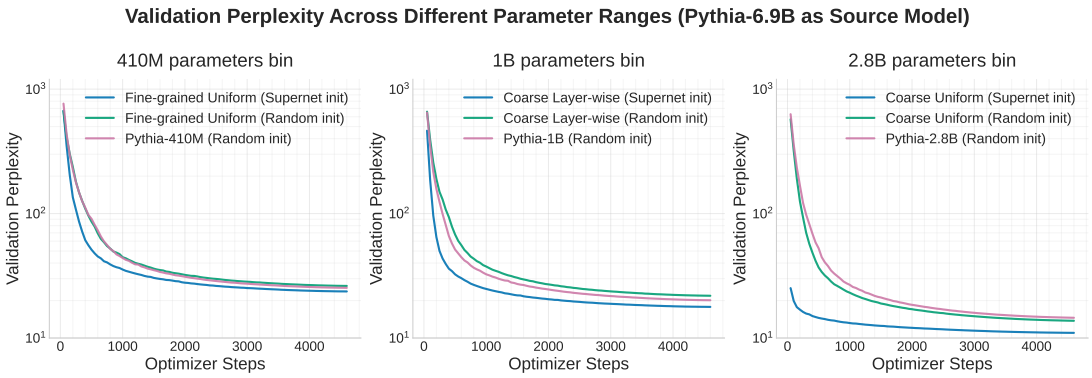


Figure 2: **Pretraining** Validation perplexity of the best sub-networks from each search space and the bin-center Pythia models (410M, 1B, 2.8B), all trained for 10B tokens with cross-entropy loss. Sub-networks are extracted from the Pythia-6.9B base model.

parameters, `whittle` operates directly on the model's pretrained checkpoint. In our experiments, we consider teacher models with more than 6 billion parameters.

We define *parameter bins* that prescribe the target size range for extracted sub-networks, which is used to guide sub-network selection and evaluation throughout our experiments. Concretely, $\mathcal{B} = [S_{\min}, S_{\max}]$, defines the range of acceptable model sizes (e.g., $S_{\min}$ = 1B, $S_{\max}$ = 2B) for a given parameter bin. We partition the overall parameter space into K disjoint bins $\{\mathcal{B}_1, \ldots, \mathcal{B}_K\}$ as follows:

$$\mathcal{S} = \{\hat{\boldsymbol{\theta}} : |\hat{\boldsymbol{\theta}}| \in [S_{\min}^{(i)}, S_{\max}^{(i)}]\}_{i=1}^{K}$$

where each bin corresponds to a contiguous range of parameter counts. The extracted sub-networks $\hat{\boldsymbol{\theta}} \subset \boldsymbol{\theta}$ are compared only with other models in the same parameter bin. This stratification also ensures balanced coverage across different model sizes. In our experiments, we consider three parameter bins ($K = 3$) for each model family.

3.2 Search Spaces

The API exposed by `whittle` can be used to extract sub-networks from the base model with a given architectural configuration. This configuration may be highly granular, down to individual neurons (*non-contiguous* mode), or more coarse-grained, where the user specifies only the sizes of architectural components such as the embedding dimension or the number of heads in a query group (*contiguous* mode). For our experiments, we define four search spaces ranging from the least to the most granular. This section describes these search spaces.

`whittle` operates on standard dense transformer checkpoints, which we denote by a teacher model T with L layers and embedding dimension E. Each layer comprises a causal multi-head self-attention block with H attention heads of dimension H_s, followed by an MLP block with intermediate dimension D. For brevity, we consider the multi-head attention setting here, though the library is compatible with multi-query and group-query attention variants.

A sub-network S is specified by selecting the number of layers $l \in \{1, \ldots, L\}$, embedding dimension $e \in \{1, \ldots, E\}$, number of attention heads $h \in \{1, \ldots, H\}$, head dimension $h_s \in \{1, \ldots, H_s\}$, and MLP intermediate size $d \in \{1, \ldots, D\}$. By varying how these dimensions are selected and shared across layers, `whittle` enables a range of sub-network granularities. In our experiments, we implement four search space configurations that differ along two axes: *coarse* versus *fine-grained* selection, and *uniform* versus *layer-wise* parameterization.

In a **coarse** search space, we always select the *first n* entries from the corresponding components of the teacher T to construct the sub-network. For example, selecting h attention heads corresponds to taking the first h heads out of the H heads in T. Likewise, choosing a smaller embedding

dimension e corresponds to taking the first e elements of the embedding vector. In **fine-grained** search spaces, on the other hand, we select a subset of size n by sampling *indices* from the teacher's components. For instance, selecting h attention heads corresponds to sampling h distinct heads from the H heads available in T (without replacement). In **uniform** search spaces, the same configuration (h, h_s, d) for heads, head size and intermediate MLP size, is applied across all layers. That is, every layer uses the same number of heads, query groups, head dimension, and MLP intermediate size. **Layer-wise** search spaces, however, allow each layer to have its own configuration, removing the uniformity constraint. The cross product of these two axes yields four search spaces: *coarse uniform* (the same first-n selection applied across all layers), *coarse layer-wise* (per-layer first-n selections), *fine-grained uniform* (the same index-sampled subset applied across all layers), and *fine-grained layer-wise* (per-layer index sampling). Their sizes range from linear in L for *coarse uniform* to exponential in both width and depth for *fine-grained layer-wise*; full size formulas are given in Appendix H.

3.3 Search Methods

To explore a given search space, we use three black-box search methods: evolutionary search, random search, and Tree-structured Parzen Estimator (TPE) [Bergstra et al., 2011]. All three methods enforce the parameter constraints subject to the parameter bin in which the search is carried out. The details of these methods can be found in Appendix B.

4 Experiments

We run experiments along two axes of flexibility offered by `whittle` . The first is along the axis of search spaces, introduced in Section 3.2. The next is along the axis of search methods, introduced in Section 3.3. To evaluate the search spaces, we use the 6.9B and 12B variants of the Pythia model family [Biderman et al., 2023] as the base models to search (Section 4.1) using evolutionary search. To evaluate the search methods, we use Qwen3-8B [Yang et al., 2025] as the base model, and compare random, evolutionary, and TPE search (Figure 21 in the Appendix).

Search Space Definitions. Table 1 summarizes the sub-network search space configurations supported by `whittle` for the Pythia-6.9B, Pythia-12B, and Qwen3-8B models, specifying the choices for each transformer dimension. In the *layer-wise* configurations, the attention and MLP dimensions (h, h_s, and d) may vary independently across layers. The *fine-grained* configurations further enable neuron-level sampling within each dimension. The resulting sizes of these search spaces are reported in Table 21 (Appendix H).

Base Model	e	h	h_s	l	d	g
`EleutherAI/pythia-6.9b`	$[1, 4096]$	$[1, 32]$	$\{4, 6, \ldots, 128\}$	$[1, 32]$	$[1, 16384]$	-
`EleutherAI/pythia-12b`	$[1, 5120]$	$[1, 40]$	$\{4, 6, \ldots, 128\}$	$[1, 36]$	$[1, 20480]$	-
`Qwen/Qwen3-8B-Base`	$[1, 4096]$	$[1, 32]$	$\{4, 6, \ldots, 128\}$	$[1, 36]$	$[1, 12288]$	$[1, 8]$

Table 1: Search space configurations for different model families. Here, e denotes embedding dimension, h the number of attention heads, h_s the head size, l the number of layers, d the MLP dimension, and g the number of query groups in grouped-query attention.

4.1 Experiments on Pythia Models

The experiments that use the Pythia base models first **search** for the sub-networks using evolutionary search, and then use **pretraining** and **knowledge distillation** to train the best sub-networks obtained in each of the three parameter bins. Sections 4.1.1, 4.1.2 and 4.1.3 discuss search, pretraining and distillation, respectively.

4.1.1 Search. We use evolutionary search (Algorithm 2, Appendix B.2) to explore sub-network configurations for the Pythia-6.9B and Pythia-12B models across the four search spaces: *coarse uniform*, *coarse layer-wise*, *fine-grained uniform*, and *fine-grained layer-wise* (Section 3.2). Figure 5 summarizes

the best validation perplexity returned by the search module for each search space configuration, illustrating the effect of search-space granularity on the quality of extracted sub-networks. See Table 10 for more details on the search parameters used to find optimal architectures using perplexity on `wikitext` [Merity et al., 2017].

4.1.2 Pretraining of Sub-networks. We use `whittle`'s pretraining pipeline to evaluate extracted sub-networks on the *Nemotron-CC* dataset [Su et al., 2025]. For each parameter bin and supported search space, the library first executes a set of low-fidelity pretraining runs with a 2B-token budget to identify promising sub-network configurations. This stage evaluates one candidate per bin across the four search space granularities, resulting in $3 \times 4 = 12$ low-fidelity runs. Based on these results, we select the highest-performing sub-network from each bin and perform higher-fidelity pretraining with a 10B-token budget on *Nemotron-CC*. All models are trained using the standard next-token prediction objective with cross-entropy loss.

Results and Discussion. Figure 2 presents the pretraining results. We compare pretraining of the extracted best sub-network (*Supernet-init*) against two baselines: (i) *Random-init*, where the same architecture is trained with random weight initialization and a 10B-token budget, and (ii) the original Pythia model (center of the bin), also trained with random initialization and the same budget. Across parameter bins, initializing from the supernet yields consistent improvements in validation perplexity. Notably, the gains are most pronounced for `bin-3`, indicating that supernet initialization is particularly beneficial in higher-parameter regimes, where our model achieved the same validation perplexity with **5.16**× fewer FLOPs. Results for a token-budget of 100B can be found in Appendix J.

4.1.3 Distillation of Sub-networks. `whittle` supports knowledge distillation as part of its training pipeline to accelerate convergence of extracted sub-networks. In our experiments, we distill from the corresponding `Pythia-6.9B` and `Pythia-12B` teacher models using a weighted combination of forward KL divergence and cross-entropy loss, with weights 0.8 and 0.2, respectively (Equation 1). For computational efficiency, we apply top-k logit distillation with $k = 1024$ and a distillation temperature of 0.9. Distillation is performed on the best-performing architecture from each parameter bin, as identified by low-fidelity pretraining with a 2B-token budget (Section 4.1.2), followed by higher-fidelity training with a 10B-token budget on *Nemotron-CC*. For each extracted sub-network, we use the same base model from which the sub-network was derived as the teacher during distillation.

Results and Discussion. Figure 20 in the Appendix illustrates the effect of distillation. We find that distillation consistently improves perplexity in both `bin-1` and `bin-2`. We also report the performance of distilled models on downstream tasks in Table 20 in Appendix E.

4.1.4 Evaluation on Downstream Tasks. Using `whittle`'s evaluation functionality, we evaluate our pretrained and distilled sub-networks on different commonsense and question-answering tasks. Specifically, we evaluate 0-shot performance on `copa`, `lambada_openai`, and `winogrande`, 5-shot performance on `MMLU`, and 10-shot performance on `arc-easy`, `arc_challenge`, `piqa`, and `hellaswag`. We report accuracy for `copa`, `lambada_openai`, `winogrande`, `MMLU` and length normalized accuracy for `piqa`, `arc_easy`, `arc_challenge` and `hellaswag`. `whittle` provides a seamless API integration with lm-eval-harness[2] for evaluating models on downstream tasks.

Results and Discussion. Table 2 reports average downstream accuracies for our best sub-networks in each parameter bin pretrained with a 10B-token budget. For comparison, we also include Pythia-410M, 1B, and 2.8B models trained with the same budget. Across all bins, `Supernet-init` outperforms both `Random-init` (for the same extracted architecture) and the original Pythia architectures (bin centers). Furthermore, sub-networks extracted from the smaller base model (Pythia-6.9B)

[2] `https://github.com/EleutherAI/lm-evaluation-harness`

Base Model	Initialization	#Params	COPA	Lambada-OpenAI	Winogrande	MMLU	PIQA	ARC-challenge	ARC-easy	HellaSwag	Avg-acc	PPL-Nemotron-cc
Pythia-6.9B	Random Init	389M	59.00	18.51	51.14	26.43	63.87	24.74	46.80	31.28	40.22	26.20
	Supernet Init	389M	61.00	24.02	51.54	26.35	65.34	24.57	51.38	33.11	**42.16**	23.66
Pythia-12B	Random Init	407M	57.00	14.87	50.67	26.33	61.64	23.97	42.47	29.76	38.34	29.57
	Supernet Init	407M	63.00	18.37	52.09	25.99	62.73	23.55	46.42	30.91	40.38	27.33
Pythia-410M*	Random Init	405M	62.00	19.54	50.67	25.54	64.14	24.57	47.35	32.70	40.81	25.29
Pythia-6.9B	Random Init	1.04B	64.00	23.36	51.54	26.62	65.78	27.13	51.80	36.83	40.38	21.84
	Supernet Init	1.04B	66.00	38.52	51.46	26.09	69.26	30.12	63.51	45.20	**48.77**	17.77
Pythia-12B	Random Init	1.04B	63.00	23.33	50.51	26.14	66.76	26.36	53.74	36.65	43.31	21.21
	Supernet Init	1.04B	64.00	27.56	51.77	26.19	66.54	26.45	53.96	36.42	44.11	20.77
Pythia-1B*	Random Init	1.01B	64.00	25.67	52.41	25.20	66.00	28.24	56.14	38.23	44.49	20.11
Pythia-6.9B	Random Init	2.91B	61.00	26.49	52.10	26.39	67.74	28.33	57.83	41.12	45.13	13.75
	Supernet Init	2.91B	66.00	50.16	56.91	26.45	72.69	33.87	67.09	53.40	**53.32**	10.99
Pythia-12B	Random Init	2.91B	67.00	27.32	50.36	25.25	67.85	27.65	57.79	40.58	45.48	13.26
	Supernet Init	2.91B	69.00	41.76	51.46	26.20	70.57	28.67	61.99	45.98	49.45	11.71
Pythia-2.8B*	Random Init	2.78B	68.00	24.51	53.03	25.74	67.68	25.34	46.67	39.11	43.76	14.54

Table 2: Evaluation of sub-networks extracted from Pythia-6.9B and Pythia-12B after pretraining on 10B tokens. A Pythia model of comparable size is also trained on the same budget with random initialization to serve as a baseline (indicated with $*$). Reported numbers are metrics as defined in Section 4.1.4 (%).

consistently outperform those from the larger base (Pythia-12B). We present results of our distilled models on downstream tasks in Table 20.

4.2 Experiments on Qwen3 Models

Using Qwen3-8B-Base as the base model, we compare three different search methods: random search, evolutionary search, and TPE. We evaluate these methods on both the original base model checkpoints and checkpoints whose weights have been sorted using Minitron importance. Based on the results from the Pythia model experiments, we fix *coarse-uniform* as the search space that we use.

4.2.1 Search. To compare the three search methods, we give them a time budget of 8 hours each. For Qwen3-8B-Base, the three parameter bins are as follows: 714M–789M (`bin-1`), 1.6B–2.4B (`bin-2`), and 3.6B–4.5B (`bin-3`). Following the implementation of models in `litgpt`, we count both the embedding and language model head weights separately in our parameter totals, even though Qwen3 ties these weights.

Results and Discussion. A comparison of search methods is shown in Figure 21. In all three bins, evolutionary search delivers the best mean performance of the incumbent sub-network. The same information is shown in Figure 3, but grouped by the usage of importance sorting in the base checkpoints. Importance sorting is seen to be extremely effective in the smallest parameter bin. Interestingly, this trend reverses in the largest bin.

Method	Init.	#Params (B)	ARC-Easy	BoolQ	CSQA	COPA	PIQA	Avg
Qwen3-0.6B-Base	Random	0.752	29.25	37.83	**20.15**	**65.00**	53.21	41.09
Evolution-Importance Sorting	Random	0.771	28.58	38.23	19.57	56.00	52.72	39.02
LLM-Pruner	Supernet	0.756	31.86	40.67	**20.15**	55.00	**55.33**	40.60
Evolution-Importance Sorting	Supernet	0.771	**32.62**	**46.18**	18.35	58.00	53.92	**41.81**

Table 3: Evaluation of Bin 1 sub-networks extracted from Qwen3-8B-Base on downstream tasks. Bold denotes the best result per column.

4.2.2 Pretraining of sub-networks. The best sub-network, that was obtained from the search phase of the smallest bin, was then pretrained with a token budget of 2 billion. We compare the extracted sub-network with three baselines: (1) Qwen3-0.6B (an official Qwen3 architecture), (2) the extracted sub-network with randomly initialized weights, (3) a sub-network discovered by `LLM-Pruner` [Ma et al., 2023] (see appendix C.1 for details) in the same parameter bin. All models were trained with the same hyper-parameters (see Table 11) and the same token budget. The results, shown in Figure

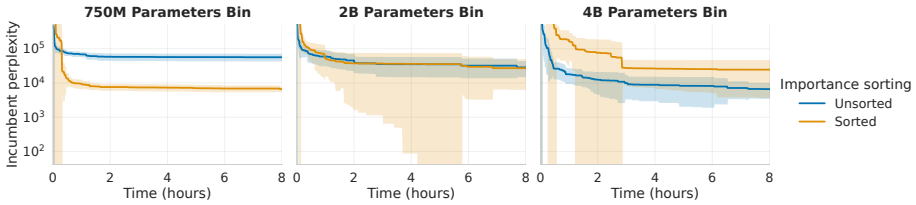


Figure 3: **Comparison of importance sorting on Qwen3**-Perplexities of incumbent sub-networks from Qwen3, grouped by importance sorting method (mean ± std over three trials).

22, clearly indicate that the extracted sub-network with weights inherited from the base model beat both Qwen3-0.6B and the randomly initialized sub-network. It also beats the model discovered by `LLM-Pruner`, but by a smaller margin.

4.2.3 Evaluation on Downstream Tasks. We evaluate pretrained architectures from Bin 1 of Qwen3-8B on five downstream tasks: `copa` (0-shot), `arc-easy` (10-shot), `piqa` (10-shot), `commonsense_qa` (10-shot), and `boolq` (10-shot), following the evaluation protocol of Li et al. [2024].

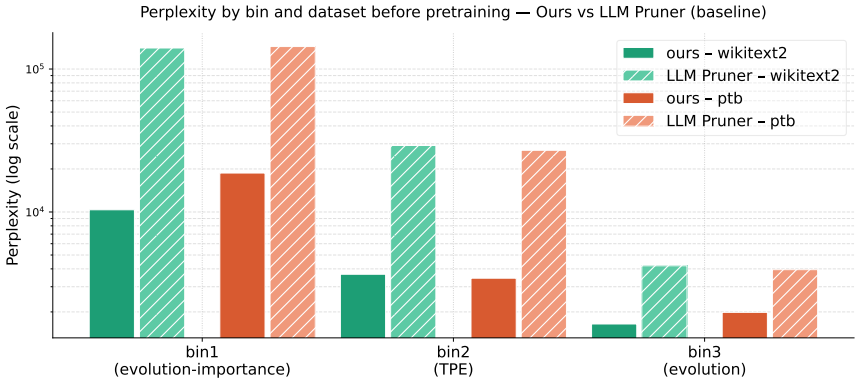


Figure 4: Comparison of perplexity on WikiText-2 and Penn Treebank (PTB) datasets for `bin-1`, `bin-2`, and `bin-3` of Qwen3-8B-Base.

Results and Discussion. Figure 4, shows the best perplexity achieved in every bin before pretraining, in comparison to LLM Pruner. Table 3 reports downstream accuracies for Bin 1 sub-networks trained on 2B tokens. Evolution-Importance Sorting with supernet initialization achieves the best average accuracy (41.81%), outperforming LLM-Pruner (40.60%) and the Qwen3-0.6B baseline (41.09%). Notably, random initialization degrades performance below the baseline (39.02%), confirming that supernet-inherited weights are essential to the gains of our method.

5 Conclusion

We present `whittle`, a framework for principled extraction and initialization of small language models (SLMs) from a pre-trained teacher. `whittle` accelerates SLM pre-training by up to 5.16× over same-size baselines, using a constrained evolutionary search to identify optimal sub-networks based on validation performance. We further show that search space granularity matters: coarser spaces favor larger SLM variants, while finer-grained spaces benefit smaller ones. By reducing pre-training cost, `whittle` lowers the barrier to entry and encourages model reuse, with potential gains in accessibility and energy efficiency, while inheriting the limitations of the teacher model. Future work includes deriving scaling laws for initialization strategies and investigating the effect of teacher model choice, including whether multilingual teachers benefit monolingual students, across domain-specific settings.

Acknowledgments

Frank Hutter acknowledges the financial support of the Hector Foundation. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation

programme under GA No 952215. We acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 539134284, through EFRE (FEIH_2698644) and the state of Baden-Württemberg. This work has been supported by LLMs4EU "Large Language Models for the European Union" project, funded by the European Union through the Digital Europe Programme under the grant agreement 101198470. This research was partially funded by the European Commission under the grant No. 10119523 (OpenEuroLLM). The authors gratefully acknowledge the computing time made available to them on the high-performance computer at the NHR Center of TU Dresden. This center is jointly supported by the Federal Ministry of Research, Technology and Space of Germany and the state governments participating in the NHR. The authors gratefully acknowledge the computing time provided on the high-performance computer HoreKa by the National High-Performance Computing Center at KIT (NHR@KIT). This center is jointly supported by the Federal Ministry of Education and Research and the Ministry of Science, Research and the Arts of Baden-Württemberg, as part of the National High-Performance Computing (NHR) joint funding program (https://www.nhr-verein.de/en/our-partners). HoreKa is partly funded by the German Research Foundation (DFG).



References

AI, L. (2023). Litgpt. `https://github.com/Lightning-AI/litgpt`.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). Gqa: Training generalized multi-query transformer models from multi-head checkpoints.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., Lochner, J., Fahlgren, C., Nguyen, X., Fourrier, C., Burtenshaw, B., Larcher, H., Zhao, H., Zakka, C., Morlon, M., Raffel, C., von Werra, L., and Wolf, T. (2025). Smollm2: When smol goes big – data-centric training of a small language model. In *Second Conference on Language Modeling*.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, S., Raff, E., et al. (2023). Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning (ICML'23)*.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR'20)*.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. (2020). The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Fang, G., Ma, X., Song, M., Mi, M. B., and Wang, X. (2023). Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16091–16101.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*.

Frantar, E. and Alistarh, D. (2023). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Gemma Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al. (2025). Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Gemma Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. (2024). Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv:1503.02531 [stat.ML]*.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2020). Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 598–605.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*.

Li, J., Fang, A., Smyrnis, G., M. Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., et al. (2024). Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282.

Liang, C., Zuo, S., Chen, M., Jiang, H., Liu, X., He, P., Zhao, T., and Chen, W. (2021). Super tickets in pre-trained language models: From model compression to improving generalization. In *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 6524–6538.

Liu, A. H., Khandelwal, K., Subramanian, S., Jouault, V., Rastogi, A., Sadé, A., Jeffares, A., Jiang, A., A.Cahill, Gavaudan, A., Sablayrolles, A., Héliou, A., You, A., Ehrenberg, A., Lo, A., Eliseev, A., Calvi, A., Sooriyarachchi, A., et al. (2026). Ministral 3.

Ma, X., Fang, G., and Wang, X. (2023). Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. (2020). Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Meta AI (2024). LLaMA 3.2: Revolutionizing edge ai and vision with open, customizable models. `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`. Accessed: 2025-08-20.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., and Kautz, J. (2019). Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

Muralidharan, S., Turuvekere Sreenivas, S., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. (2024). Compact language models via pruning and knowledge distillation. In *Advances in Neural Information Processing Systems*, volume 37, pages 41076–41102.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In *International Conference of Learning Representations*.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Proceedings of Thirty-third Conference on Neural Information Processing Systems (NIPS2019)*.

Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

Su, D., Kong, K., Lin, Y., Jennings, J., Norick, B., Kliegl, M., Patwary, M., Shoeybi, M., and Catanzaro, B. (2025). Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2459–2475.

Taylor, A., Marcus, M., and Santorini, B. (2003). The penn treebank: an overview. *Treebanks: Building and using parsed corpora*, pages 5–22.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS'17)*.

Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., and Han, S. (2020). Hat: Hardware-aware transformers for efficient natural language processing. In *Annual Meeting of the Association for Computational Linguistics*.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., and Hutter, F. (2023). Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*.

Xu, X., Li, M., Tao, C., Shen, T., Cheng, R., Li, J., Xu, C., Tao, D., and Zhou, T. (2024). A survey on knowledge distillation of large language models. *CoRR*, abs/2402.13116.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. (2025). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhang, B. and Sennrich, R. (2019). Root mean square layer normalization. In *Proceedings of the 32th International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. (2021). Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*.

A Related Work

Model Pruning and Sub-Networks. Pruning is an approach for compressing neural networks by removing redundant parameters while preserving performance [LeCun et al., 1990, Han et al., 2016]. While early work focused on unstructured magnitude pruning, more recent methods emphasize structured and semi-structured sparsity to improve practical efficiency on modern hardware [Li et al., 2017, Zhou et al., 2021, Ma et al., 2023, Frantar and Alistarh, 2023]. The Lottery Ticket Hypothesis (LTH) [Frankle and Carbin, 2018] further showed that large networks contain sub-networks that can train in isolation to match full-model performance, motivating extensive follow-up work on their properties and scalability, including in pretrained language models [Frankle et al., 2019, Malach et al., 2020, Chen et al., 2020, Liang et al., 2021]. A central challenge across pruning methods is the design of importance criteria that scale to billion-parameter models [Molchanov et al., 2019, Frantar and Alistarh, 2023]. Our work builds on this line by providing a *heuristic-free* practical framework for extracting high-quality structured sub-networks from pretrained LLM checkpoints.

`torch-pruning` [Fang et al., 2023] is a library that is particularly relevant in a discussion about prior works related to `whittle`. `torch-pruning` prunes a model by representing it as a dependency graph. By tracking which parameters are coupled, it applies importance criteria to remove structure and shrink the model to a target pruning ratio. Like `whittle`, it also exposes lower-level APIs to prune specific channels by index. The two libraries differ in scope and mechanism. `torch-pruning` is architecture-agnostic: it discovers dependencies automatically for arbitrary PyTorch models and prunes them in-place. `whittle` instead targets language models, is tightly coupled with `litgpt`, and exposes APIs shaped around the transformer architecture. Because the architecture and the resulting coupling between prunable dimensions are fixed and known a priori, `whittle` prunes without constructing a dependency graph. Crucially, `whittle` represents a sub-network as a sliced (index-selected) view of a weight-shared super-network, so many candidates can be evaluated cheaply from a single set of shared weights; this makes it well suited for NAS methods that treat the model as a search space to optimize. Further, `whittle` also extends the train and distill functions already available in `litgpt`, so users can train their models using the same workflows.

Another relevant work is `sparseml`[3], a library for compressing neural networks using pruning, quantization, and distillation. It contains a large collection of pruning methods that can be used out of the box to sparsify models; these are unified by a declarative YAML *recipe* system. Pruned weights are masked in place rather than removed, so the compressed model retains the shape of the original. Both libraries are extensible, but at different levels: `sparseml` exposes a modifier plugin interface for creating new pruning criteria and schedules, whereas `whittle` is built around sub-network extraction and is therefore NAS-friendly by construction. With `whittle`, a user may deploy their own algorithms to identify the neurons to prune, then use the library to slice those neurons out of a super-network (which is a reversible operation) and extract the smaller dense model. `sparseml`, however, does not expose APIs to control pruning at such a granular level. Rather, it focuses on selecting neurons by norm-based ranking. The library has been archived and is no longer maintained.

Knowledge Distillation. Knowledge distillation (KD) compresses large models by transferring information from a teacher to a smaller student, often improving both efficiency and convergence [Hinton et al., 2015, Xu et al., 2024]. For language models, KD is commonly applied at the output level using logit matching with KL divergence, frequently combined with top-k or top-p truncation [Sanh et al., 2019, Gemma Team et al., 2024], or at the representation level by aligning hidden states [Romero et al., 2015, Jiao et al., 2020]. These approaches highlight KD as a flexible and effective tool for training compact language models, which we adopt as a core component of our pipeline.

[3] https://github.com/neuralmagic/sparseml

Neural Architecture Search. Neural Architecture Search (NAS) explores large design spaces using black-box optimization or differentiable search [Elsken et al., 2019, White et al., 2023]. While effective at smaller scales, state-of-the-art NAS methods for large models rely on training a super-network that contains many candidate sub-networks [Cai et al., 2020, Wang et al., 2020]. In contrast to pruning-based approaches, NAS selects sub-networks via black-box optimization. In this work, we instead leverage existing pretrained LLM checkpoints to search directly over sub-networks, avoiding super-network training and focusing on efficient warm-start initialization rather than standalone architecture design.

Practical Software Tools for Small Language Model Extraction. Recent language model families are released at multiple scales to balance performance and efficiency [Biderman et al., 2023]. Training each scale independently is computationally costly, motivating approaches that derive small language models (SLMs) from large pretrained models via pruning and distillation [Muralidharan et al., 2024, Meta AI, 2024, Gemma Team et al., 2025, Liu et al., 2026]. Despite this progress, existing solutions are often tightly coupled to specific model families, lack modularity, and are not released as open-source frameworks [Muralidharan et al., 2024, Meta AI, 2024, Gemma Team et al., 2025, Liu et al., 2026]. Our work addresses this gap by introducing a library-centered approach that provides a lightweight wrapper around existing open-source LLM frameworks, such as Hugging Face[4] and `litgpt`[5], unifying sub-network extraction, search, and distillation within an open-source and extensible framework.

B Search Methods

In this section, we discuss the three search methods considered in this paper: random search (Section B.1), evolutionary search (Section B.2), and Bayesian search with Tree-structured Parzen estimators (Section B.3).

B.1 Random Search

Within each bin $\mathcal{B}_i$, we perform a random search over candidate sub-network architectures $\mathcal{A}(\boldsymbol{\theta})$. At each iteration, candidate architectures are sampled uniformly at random from the search space and evaluated according to a fitness function $f(\mathcal{A})$. To enforce the bin constraint, we apply *rejection sampling*:

$$\mathcal{A} \leftarrow \begin{cases} \mathcal{A}, & \text{if } |\boldsymbol{\theta}_{\mathcal{A}}| \in \mathcal{B}_i, \\ \text{reject}, & \text{otherwise.} \end{cases}$$

Only candidates satisfying $|\boldsymbol{\theta}_{\mathcal{A}}| \in \mathcal{B}_i$ are retained for evaluation.

Selection. After exhausting the sampling budget, we return the set of small language models (one for each bin)

$$\mathcal{A}_i^* = \arg \max_{|\boldsymbol{\theta}_{\mathcal{A}}| \in \mathcal{B}_i} f(\mathcal{A}),$$

that achieve the most favorable initialization for subsequent pretraining or distillation. Here, f is the negative perplexity of the architecture on a small batch of data.

The random search is summarized in Algorithm 1. For each bin, `whittle` draws N candidate sub-networks independently and uniformly from the constrained search space and evaluates each using validation perplexity. After T iterations, the search module returns the highest-performing sub-network for each parameter bin. While random search is less sample-efficient than evolutionary or TPE-based methods, it serves as a strong and reproducible baseline, particularly in low-budget regimes where the overhead of surrogate model fitting may outweigh its benefits.

[4] `https://huggingface.co`

[5] `https://github.com/Lightning-AI/litgpt`

Algorithm 1 Bin-Constrained Random Search

Input: arch. space $\mathcal{S}$; bins $\{[L_b, U_b]\}_{b=1}^{B}$; iterations T
Helpers: Params(s) = param. count; ppl(s) = perplexity
for $b = 1$ **to** B **do**
    $\mathcal{S}_b \leftarrow \{s \in \mathcal{S} \mid L_b \leq \text{Params}(s) \leq U_b\}$
    $\mathcal{H}_b \leftarrow \emptyset$
    for $t = 1$ **to** T **do**
        Sample $s \sim \mathcal{S}_b$ uniformly at random
        Evaluate ppl(s)
        $\mathcal{H}_b \leftarrow \mathcal{H}_b \cup \{s\}$
    end for
    Best $s_b^\star \leftarrow \arg\min_{s \in \mathcal{H}_b} \text{ppl}(s)$
end for
Output: $\{s_b^\star\}_{b=1}^{B}$

B.2 Evolutionary Search

For each bin $\mathcal{B}_i$, we perform an evolutionary search over candidate sub-network architectures $\mathcal{A}(\boldsymbol{\theta})$. Rejection sampling is used to enforce the bin constraint, and the set of small language models that achieve the best initialization is returned, as in random search (Section B.1). These selected SLMs represent the optimal sub-network architectures within the specified parameter range.

The evolutionary search is summarized in Algorithm 2. For each bin, the library initializes a population of candidate sub-networks drawn from the corresponding constrained search space and evaluates them using validation perplexity. At each epoch, `whittle` retains the top-k candidates and applies configurable genetic operators, including **mutation** and **crossover**, to generate new candidates that respect the bin constraints. Additional random sampling is used to promote exploration. The population for the next epoch is formed by selecting the top N candidates from the union of retained, generated, and randomly sampled sub-networks. After T epochs, the search module returns the highest-performing sub-network for each parameter bin. The **mutation** operation introduces random perturbations to sub-network dimensions under bin constraints, whereas the **crossover** operation generates new candidates by recombining dimensions from selected parent sub-networks. We discuss them next.

B.2.1 Mutation. Given a candidate architecture $\mathcal{A}$, we define a mutation operator $\mu(\mathcal{A})$ that perturbs one architectural dimension at a time. Specifically, we uniformly sample a dimension

$$x \in \{l, e, h, g, d, h_s\},$$

where l denotes the number of layers, e the embedding dimension, h the number of attention heads, g the number of query groups, d the intermediate (feedforward) dimension, and h_s the per-head size.

Mutation in layer-wise search space.. In *layer-wise* search spaces, architectural attributes (h, g, d, h_s) are defined independently for each layer. A mutation of the layer count $l \rightarrow l'$ is handled as follows:

$$\mathcal{A}' = \begin{cases} \mathcal{A} \cup \text{newly sampled}(l' - l)\text{layers}, & \text{if } l' > l, \\ \mathcal{A} \setminus \text{last } (l - l') \text{ layers}, & \text{if } l' < l \end{cases}$$

If $x \in \{h, g, d, h_s\}$, we first sample a layer index $i \sim \text{Uniform}\{1, \ldots, l\}$, then resample the chosen dimension for that layer:

$$x_i' \sim \text{Uniform}\{\text{choices}(x)\}$$

Where $\text{choices}(x)$ denotes the valid choices for an architectural attribute x defined by a search space. All other architectural parameters remain fixed.

Mutation in fine-grained search space.. In the *fine-grained* setting, mutations operate at the neuron level. For instance, mutating the embedding dimension $e \to e'$ corresponds to

$$\begin{aligned} &\text{if } e' > e: \text{ sample } (e' - e) \text{ new neurons,} \\ &\text{if } e' < e: \text{ prune the last } (e - e') \text{ neurons.} \end{aligned} \tag{2}$$

This formulation enables smooth exploration of architectures across both coarse (layer-wise) and fine-grained structural variations, while maintaining consistency with the model size constraint.

B.2.2 Crossover. To produce a child architecture from two parents, P_1 and P_2, we apply a crossover operator $\chi(P_1, P_2)$. We first require both parents to share the same number of layers:

$$l_{P_1} = l_{P_2} = l$$

Let the architectural dimensions of each parent be

$$P_1 = (e_1, h_1, g_1, h_{s,1}, d_1), \quad P_2 = (e_2, h_2, g_2, h_{s,2}, d_2),$$

where e, h, g, h_s, and d denote the embedding dimension, number of attention heads, number of query groups, head size, and intermediate dimension, respectively.

A child architecture c is then generated by independently inheriting each dimension from one of the two parents (w.p. stands for with probability):

$$x_c = \begin{cases} x_1, & \text{w.p. } 0.5, \\ x_2, & \text{w.p. } 0.5, \end{cases} \qquad x \in \{e, h, g, h_s, d\}. \tag{3}$$

For example, a valid crossover outcome might be

$$c = (e_2, h_1, g_2, h_{s,2}, d_1)$$

This independent dimension-wise crossover enables fine-grained recombination of architectural traits while preserving structural compatibility (e.g., layer count consistency) between parents.

B.3 Tree-structured Parzen Estimator Search

We perform a TPE-guided search [Bergstra et al., 2011] over candidate sub-network architectures $\mathcal{A}(\boldsymbol{\theta})$, as implemented in Optuna [Akiba et al., 2019], within each bin $\mathcal{B}_i$. TPE-guided search uses rejection sampling and selects a set of incumbent architectures in the same way as evolutionary and random search (Section B.1).

The TPE search is summarized in Algorithm 3. TPE is a sequential model-based optimization algorithm that maintains two density estimators at each trial: $\ell(x)$, a Gaussian Mixture Model (GMM) over architectural configurations associated with good objective values, and $g(x)$, a GMM over the remaining configurations. A new candidate is proposed by maximizing the ratio $\ell(x)/g(x)$, which corresponds to maximizing the Expected Improvement (EI) under the model. For the first n_{startup} trials, candidates are drawn uniformly at random to bootstrap the density estimators before switching to TPE-guided proposals. Unlike Gaussian Process-based Bayesian optimization, TPE scales favorably to high-dimensional and mixed search spaces, making it well suited to sub-network search over discrete architectural dimensions such as embedding size, number of attention heads, MLP width, and depth. We use Optuna's `TPESampler` with its default configuration, applying the same parameter budget and bin constraints as in the evolutionary and random search baselines, enabling direct comparison across strategies.

Algorithm 2 Bin-Constrained Evolutionary Search

Input: arch. space $\mathcal{S}$; bins $\{[L_b, U_b]\}_{b=1}^{B}$; population N; elites k; epochs T; random samples r; number of offsprings λ; mutation prob. m; crossover prob. c, mutation function $\mathrm{Mut}(s)$, crossover function $\mathrm{Cross}(s, s')$
Helpers: $\mathrm{Params}(s)$ = param. count; $\mathrm{ppl}(s)$ = perplexity
$\textsc{Constrain}(x, [L, U])$: resample/repair until $\mathrm{Params}(x) \in [L, U]$
for $b = 1$ **to** B **do**
    $\mathcal{S}_b \leftarrow \{s \in \mathcal{S} \mid L_b \le \mathrm{Params}(s) \le U_b\}$
    Init population $\mathcal{P}_b^{(0)} \sim \mathcal{S}_b$
    for $t = 0$ **to** $T - 1$ **do**
        Evaluate $\mathrm{ppl}(s)$ for $s \in \mathcal{P}_b^{(t)}$
        Select elites $\mathcal{E}_b^{(t)} = \arg\min^k \mathrm{ppl}(s)$
        Mutants $\mathcal{O}_{\mathrm{mut}} \leftarrow \textsc{Constrain}(\mathrm{Mut}(s), [L_b, U_b])$, $s \sim \mathcal{E}_b^{(t)}$, size λ (with prob. m)
        Crossovers $\mathcal{O}_{\mathrm{cross}} \leftarrow \textsc{Constrain}(\mathrm{Cross}(s, s'), [L_b, U_b])$, $s, s' \sim \mathcal{E}_b^{(t)}$, size λ (with prob. c)
        Randoms $\mathcal{R}_b^{(t)} \sim \mathcal{S}_b$, size r
        Next pop. $\mathcal{P}_b^{(t+1)} \leftarrow \arg\min^N \mathrm{ppl}(s)$ over $\mathcal{E}_b^{(t)} \cup \mathcal{O}_{\mathrm{mut}} \cup \mathcal{O}_{\mathrm{cross}} \cup \mathcal{R}_b^{(t)}$
    end for
    Best $s_b^\star \leftarrow \arg\min_{s \in \mathcal{P}_b^{(T)}} \mathrm{ppl}(s)$
end for
Output: $\{s_b^\star\}_{b=1}^{B}$

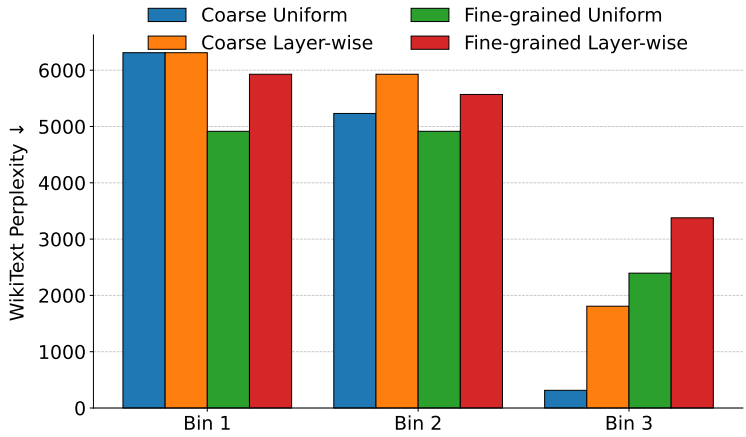


Figure 5: Best perplexity after evolutionary search on the Pythia base model for different search spaces. Note that here, the extracted models are evaluated before pretraining or distillation.

C Additional Experimental Details

C.1 LLM-Pruner

We evaluate `LLM-Pruner` [Ma et al., 2023] as a structural pruning baseline, which we apply to the Qwen3-8B base model. LLM-Pruner supports three levels of granularities for pruning, namely, *blockwise*, *layerwise*, and *channelwise*. Blockwise pruning removes attention heads and MLP neurons jointly within each transformer block, pruning the coupled structures identified by Taylor importance [Molchanov et al., 2019]. Layerwise pruning discards entire transformer layers from the end of the network, keeping only the first N layers, where N is a hyperparameter. Channelwise pruning

Algorithm 3 Bin-Constrained TPE Search

Input: arch. space $\mathcal{S}$; bins $\{[L_b, U_b]\}_{b=1}^{B}$; trials T; startup trials n_{startup}
Helpers: Params(s) = param. count; ppl(s) = perplexity
for $b = 1$ **to** B **do**
    $\mathcal{S}_b \leftarrow \{s \in \mathcal{S} \mid L_b \le \text{Params}(s) \le U_b\}$
    Initialize TPE sampler with n_{startup} random trials
    $\mathcal{H}_b \leftarrow \emptyset$
    for $t = 1$ **to** T **do**
        if $t \le n_{\text{startup}}$ **then**
            Sample $s \sim \mathcal{S}_b$ uniformly at random
        else
            Fit $\ell(x)$ on top-γ candidates and $g(x)$ on remaining candidates in $\mathcal{H}_b$
            Propose $s \leftarrow \arg\max_{x \sim \ell} \frac{\ell(x)}{g(x)}$ subject to Params(s) $\in [L_b, U_b]$
        end if
        Evaluate ppl(s)
        $\mathcal{H}_b \leftarrow \mathcal{H}_b \cup \{s\}$
    end for
    Best $s_b^\star \leftarrow \arg\min_{s \in \mathcal{H}_b} \text{ppl}(s)$
end for
Output: $\{s_b^\star\}_{b=1}^{B}$

reduces the hidden dimension uniformly across all linear projections, using Taylor importance criterion to rank and remove the channels globally.

Blockwise pruning failed to produce an architecture within the smallest bin (714M-789M), whereas layerwise pruning yielded architectures with substantially higher perplexity on both WikiText [Merity et al., 2017] and Penn Treebank [Taylor et al., 2003] than channelwise pruning. We therefore report channelwise pruning as the LLM-Pruner baseline in our experiments. The pruned models for bins 1, 2 and 3 have 755.89M, 1.8B, and 3.68B parameters, respectively.

C.2 Hyperparameter Configurations of Experiments

In Tables 4 - 12, we present the hyperparameter settings for all our experiments.

C.3 Computational Cost of the Evolutionary Search

In this section, we provide an overview of the cost overhead introduced by evolutionary search. In each bin, for every search space, we sample and evaluate a total of 5,050 subnetworks during the evolutionary search. For each candidate model, we computed the perplexity on 1,000 sequences of length 512. We approximate the average FLOP of the models in bins 1, 2 and 3 as the FLOPs of Pythia-410M, Pythia-1B, and Pythia-2.8B, since these models serve as the center of the bins. The total computational cost of the search for each search space is reported in Table 13. For comparison, the cost of pretraining the best model found in each bin on 10B tokens is as presented in Table 14. As Tables 13 and 14 indicate, the search phase consumes only a small fraction of the overall pretraining budget.

Detailed Computation of Cost Savings. We include the cost of the evolutionary search when computing the total cost savings achieved by our method. The updated FLOP-savings factors are reported below in Table 15. Additionally, we include the FLOP-savings factor considering *only* the pretraining budget in Table 16.

Furthermore, in Figure 6, we present the validation perplexity across different parameter ranges, taking the search cost into account.

Search Space	Parameter range	Hyperparameter Type		Value
Evolutionary Search Coarse	Bin 1 385M–426M	Model & Data	Model Name	pythia-12b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	8
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse Layerwise	Bin 2 961M–1.06B	Model & Data	Model Name	pythia-12b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	8
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse	Bin 3 2.64B–2.91B	Model & Data	Model Name	pythia-12b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	1.6×10^{-4}
			Min Learning Rate	1.6×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	16
			LR Warmup Steps	238
			Max Sequence Length	2048
			Seed	42

Table 4: Hyperparameters used for Pre-training the Best Performing Subnets per Bin (Parameter Range) from the Pythia-12B Model

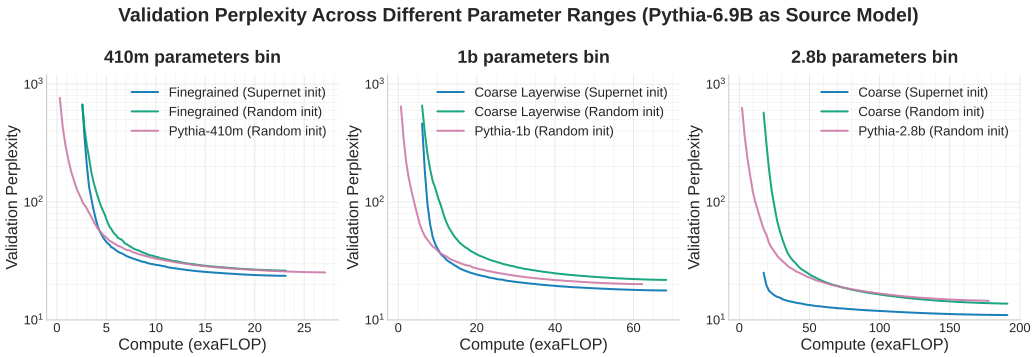


Figure 6: Validation perplexity across different parameter ranges (offset with search cost).

Search Space	Parameter range	Hyperparameter Type		Value
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Model Name	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse Layerwise	Bin 2 961M–1.06B	Model & Data	Model Name	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	4
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse	Bin 3 2.64B–2.91B	Model & Data	Model Name	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	1.6×10^{-4}
			Min Learning Rate	1.6×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	16
			LR Warmup Steps	238
			Max Sequence Length	2048
			Seed	42

Table 5: Hyperparameters used for Pre-training the Best Performing Subnets per Bin (Parameter Range) from the Pythia-6.9B Model

D Additional Details on Experimental Setup

In this section we provide additional details on our experimental setup described in Section 3.

D.1 Attention masking

The attention mechanism used in transformer blocks naturally supports sub-network extraction. In practice, this means that an attention mechanism can be masked to yield a smaller, distinct type of attention. Figure 8 provides an overview of the main variants—multi-head attention (MHA), multi-query attention (MQA), and grouped-query attention (GQA). Since GQA serves as a super-class of these mechanisms, it can be transformed into either MHA or MQA. An illustration of this transformation is shown in Figure 9.

D.2 Details on Distillation Module

Top-k Logit Distillation. We define a variant of knowledge distillation loss in `whittle` that truncates the teacher distribution to its k most salient outputs. Let $z_t, z_s \in \mathbb{R}^C$ denote the teacher and student logits, respectively, and T the temperature parameter. The teacher distribution is given by $p_t = \text{softmax}(z_t/T)$. We denote by $\mathcal{K} \subset \{1, \ldots, C\}$ either the indices of the top-k logits of z_s, or a subset sampled from p_t. The distillation loss is then defined as:

$$\mathcal{L}_{\text{top-}k} = \sum_{i \in \mathcal{K}} \text{KL}\left(\text{softmax}\left(\frac{z_t^{(i)}}{T} \right) \,\|\, \text{softmax}\left(\frac{z_s^{(i)}}{T} \right) \right). \tag{4}$$

Search Space	Parameter range	Hyperparameter Type	Value
Evolutionary Search Coarse	Bin 1 385M–426M	Model & Data	Teacher Model: pythia-12b Precision: bf16-mixed Dataset: Nemotron-CC
		Optimizer & Regularization	Optimizer: AdamW Learning Rate: 3×10^{-4} Min Learning Rate: 3×10^{-5} Weight Decay: 0.01 AdamW β_1, β_2: 0.9, 0.95 Gradient Clipping Norm: 1.0
		Distillation	α: 0.2 β: 0.8 Temperature: 0.9 Logits: Top-1024
		Training & Batching	Total Training Tokens: 10B Global Batch Size: 1056 Micro Batch Size: 2 LR Warmup Steps: 0 Max Sequence Length: 2048 Seed: 42
Evolutionary Search Coarse Layerwise	Bin 2 961M–1.06B	Model & Data	Teacher Model: pythia-12b Precision: bf16-true Dataset: Nemotron-CC
		Optimizer & Regularization	Optimizer: AdamW Learning Rate: 3×10^{-4} Min Learning Rate: 3×10^{-5} Weight Decay: 0.01 AdamW β_1, β_2: 0.9, 0.95 Gradient Clipping Norm: 1.0
		Distillation	α: 0.2 β: 0.8 Temperature: 0.9 Logits: Top-1024
		Training & Batching	Total Training Tokens: 10B Global Batch Size: 1056 Micro Batch Size: 8 LR Warmup Steps: 0 Max Sequence Length: 2048 Seed: 42
Evolutionary Search Coarse	Bin 3 2.64B–2.91B	Model & Data	Teacher Model: pythia-12b Precision: bf16-true Dataset: Nemotron-CC
		Optimizer & Regularization	Optimizer: AdamW Learning Rate: 1.6×10^{-4} Min Learning Rate: 1.6×10^{-5} Weight Decay: 0.01 AdamW β_1, β_2: 0.9, 0.95 Gradient Clipping Norm: 1.0
		Distillation	α: 0.2 β: 0.8 Temperature: 0.9 Logits: Top-1024
		Training & Batching	Total Training Tokens: 10B Global Batch Size: 1056 Micro Batch Size: 4 LR Warmup Steps: 238 Max Sequence Length: 2048 Seed: 42

Table 6: Hyperparameters used for Distillation Experiments on the Best Performing Subnets per Bin (Parameter Range) from the Pythia-12B Model

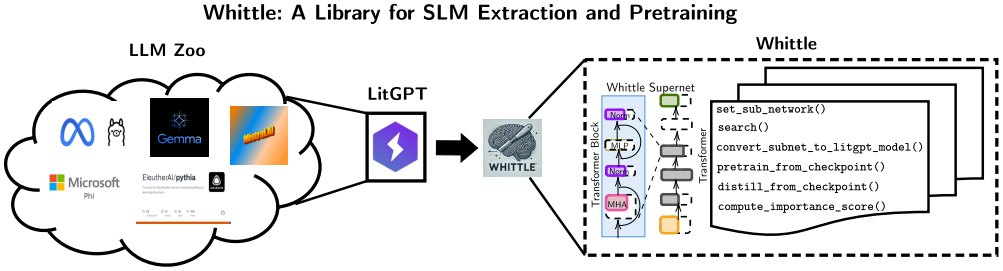


Figure 7: An overview of the `whittle`library. LitGPT supports a wide range of LLMs. `whittle`, being a light-weight layer built on top of it, is able to apply its APIs to the models supported by LitGPT.

Search Space	Parameter range	Hyperparameter Type		Value
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse Layerwise	Bin 2 961M–1.06B	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	4
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Coarse	Bin 3 2.64B–2.91B	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	1.6×10^{-4}
			Min Learning Rate	1.6×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	10B
			Global Batch Size	1056
			Micro Batch Size	4
			LR Warmup Steps	238
			Max Sequence Length	2048
			Seed	42

Table 7: Hyperparameters used for Distillation Experiments on the Best Performing Subnets per Bin (Parameter Range) from the Pythia-6.9B Model

In addition, `whittle` supports different types of distillation loss functions, defined in the Table 17.

E Additional Results

In this section we present ablations for different functionalities of `whittle`.

E.1 Ablations

In this section, we conduct ablation studies to examine the effect of three key factors in our framework: (a) the choice of search space, (b) the loss function used for distillation, (c) the performance metric employed during search.

Granularity of Search Spaces. Figure 10 illustrates the effect of varying search space granularity. We find that different bins benefit from distinct choices: for `bin-1`, *fine-grained uniform* search space is optimal; for `bin-2`, *coarse layer-wise* performs best; and for `bin-3`, *coarse uniform* yields the strongest results.

Search Space	Parameter range	Hyperparameter Type		Value
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Teacher Model	pythia-1b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	2B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	2B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42

Table 8: Hyperparameters used for Distillation Ablation Experiments on a Subnet from the Pythia-6.9B Model using varying Teacher Model Sizes

Search Space	Parameter range	Hyperparameter Type		Value
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Top-1024
		Training & Batching	Total Training Tokens	2B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42
Evolutionary Search Finegrained	Bin 1 385M–426M	Model & Data	Teacher Model	pythia-6.9b
			Precision	bf16-mixed
			Dataset	Nemotron-CC
		Optimizer & Regularization	Optimizer	AdamW
			Learning Rate	3×10^{-4}
			Min Learning Rate	3×10^{-5}
			Weight Decay	0.01
			AdamW β_1, β_2	0.9, 0.95
			Gradient Clipping Norm	1.0
		Distillation	α	0.2
			β	0.8
			Temperature	0.9
			Logits	Full
		Training & Batching	Total Training Tokens	2B
			Global Batch Size	1056
			Micro Batch Size	6
			LR Warmup Steps	0
			Max Sequence Length	2048
			Seed	42

Table 9: Hyperparameters used for Distillation Ablation Experiments on a Subnet from the Pythia-6.9B Model — Top-K vs. Full Logits

Parameter	Value
max_epochs	100
population_num	50
select_num	10
mutation_num	25
crossover_num	25
m_prob	0.2
s_prob	0.2
seq_len	512
batch_size	2
dataset	WikiText

Table 10: Search Hyperparameters for the Pythia Subnet Search for All 3 Bins - 385M–426M (bin-1), 961M–1.06B (bin-2), and 2.64B–2.91B (bin-3), for all four Search Spaces

Parameter range	Hyperparameter Type		Value
Bin 1 714M–789M	Model & Data	Model Name	Qwen3-8B-Base
		Precision	bf16-mixed
		Dataset	Nemotron-CC
	Optimizer & Regularization	Optimizer	AdamW
		Learning Rate	3×10^{-4}
		Min Learning Rate	3×10^{-5}
		Weight Decay	0.1
		AdamW β_1, β_2	0.9, 0.95
		Gradient Clipping Norm	1.0
	Training & Batching	Total Training Tokens	2B
		Global Batch Size	1056
		Micro Batch Size	4
		LR Warmup Steps	4734
		Max Sequence Length	4096
		Seed	42
Bin 2 1.638B–2.457B	Model & Data	Model Name	Qwen3-8B-Base
		Precision	bf16-mixed
		Dataset	Nemotron-CC
	Optimizer & Regularization	Optimizer	AdamW
		Learning Rate	2×10^{-4}
		Min Learning Rate	2×10^{-5}
		Weight Decay	0.1
		AdamW β_1, β_2	0.9, 0.95
		Gradient Clipping Norm	1.0
	Training & Batching	Total Training Tokens	2B
		Global Batch Size	1056
		Micro Batch Size	2
		LR Warmup Steps	9469
		Max Sequence Length	4096
		Seed	42
Bin 3 3.685B–4.504B	Model & Data	Model Name	Qwen3-8B-Base
		Precision	bf16-mixed
		Dataset	Nemotron-CC
	Optimizer & Regularization	Optimizer	AdamW
		Learning Rate	1.75×10^{-4}
		Min Learning Rate	1.75×10^{-5}
		Weight Decay	0.1
		AdamW β_1, β_2	0.9, 0.95
		Gradient Clipping Norm	1.0
	Training & Batching	Total Training Tokens	2B
		Global Batch Size	1056
		Micro Batch Size	1
		LR Warmup Steps	18939
		Max Sequence Length	4096
		Seed	42

Table 11: Hyperparameters used for Pre-training the Subnets extracted from the Qwen3-8B Model

Parameter	Evolutionary Coarse	Evolutionary Importance Sorted	Optuna Coarse
max_epochs / n_trials	100	100	50000
population_num	50	50	—
select_num	10	10	—
mutation_num	25	25	—
crossover_num	25	25	—
m_prob	0.2	0.2	—
s_prob	0.4	0.4	—
sampler	—	—	TPE
timeout	—	—	8h
seq_len	512	512	512
batch_size	2	2	2
dataset	WikiText	WikiText	WikiText

Table 12: Search Hyperparameters for the Qwen3-8B Subnet Search for All 3 Bins (Parameter Ranges)

Bin	Search Cost (exaFLOP)
1	2.3
2	5.4
3	15.4

Table 13: Cost of Evolutionary Search for different bins

Bin	Pretraining Cost (exaFLOP)
1	20.9
2	63.3
3	176.6

Table 14: Cost of pretraining the best models in different bins

Bin	FLOP Savings Factor
1	1.71×
2	1.75×
3	5.16×

Table 15: FLOP-saving factors for all bins compared to pretraining the corresponding Pythia architectures.

Bin	FLOP Savings Factor (excluding search cost)
1	2.0×
2	2.07×
3	9.32×

Table 16: FLOP-saving factors for all bins compared to pretraining the corresponding Pythia architectures (without considering the search cost).

Full logits vs. top-k logits. In our distillation experiments in Section 4, following [Gemma Team et al., 2025], we use top-k logit based distillation. Here, we ablate this choice for the distillation loss

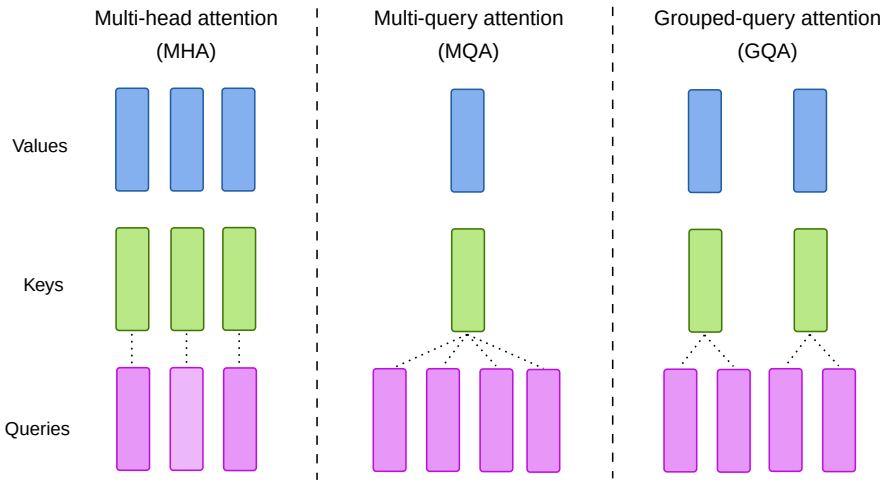


Figure 8: An illustration of the different types of attention mechanisms. In multi-head attention (MHA), each query is paired with its own key and value; in multi-query attention (MQA), multiple queries share a single key–value pair; and in grouped-query attention (GQA), multiple key–value pairs are used, with each pair serving more than one query.

Loss Type	Equation
Forward KLD	$\sum p(t) \log \frac{p(t)}{q(t)}$
Reverse KLD	$\sum q(t) \log \frac{q(t)}{p(t)}$
JS Divergence	$\frac{1}{2}\left(\sum p(t) \log \frac{2p(t)}{p(t)+q(t)} + \sum q(t) \log \frac{2q(t)}{p(t)+q(t)}\right)$
L2-Norm Distance	$\|\theta_T(x) - \theta_S(x)\|_2$
L1-Norm Distance	$\|\theta_T(x) - \theta_S(x)\|_1$

Table 17: Summary of possible forms for $\mathcal{D}$ in knowledge distillation. Here, $p(t)$ and $q(t)$ represent the teacher and student distributions, respectively, while θ_T and θ_S denote the teacher and student sub-networks respectively and $\theta_T(x)$ and $\theta_S(x)$ correspond to their respective output logits.

by comparing supervision from the full teacher distribution against a truncated variant using only the top-k logits (Figure 11). This isolates how much of the teacher's probability mass is required for effective transfer. We find that, in general, *distilling from the full-logit distribution yields a lower perplexity.*

We define our distillation loss function below.

$$\mathcal{L} = \alpha \, \mathbf{L}_{\text{CE}}(\mathbf{s}, \mathbf{y}) + \beta \sum_{i \in \mathcal{K}} \text{KL}\left(\text{softmax}\left(\frac{z_t^{(i)}}{T}\right) \,\middle\|\, \text{softmax}\left(\frac{z_s^{(i)}}{T}\right)\right). \tag{5}$$

This loss has four tunable hyperparameters:

1. α: the weight of the cross-entropy loss, which minimizes the entropy with respect to the ground-truth logits.

2. β: the weight of the KL-divergence term between the teacher and student logits.

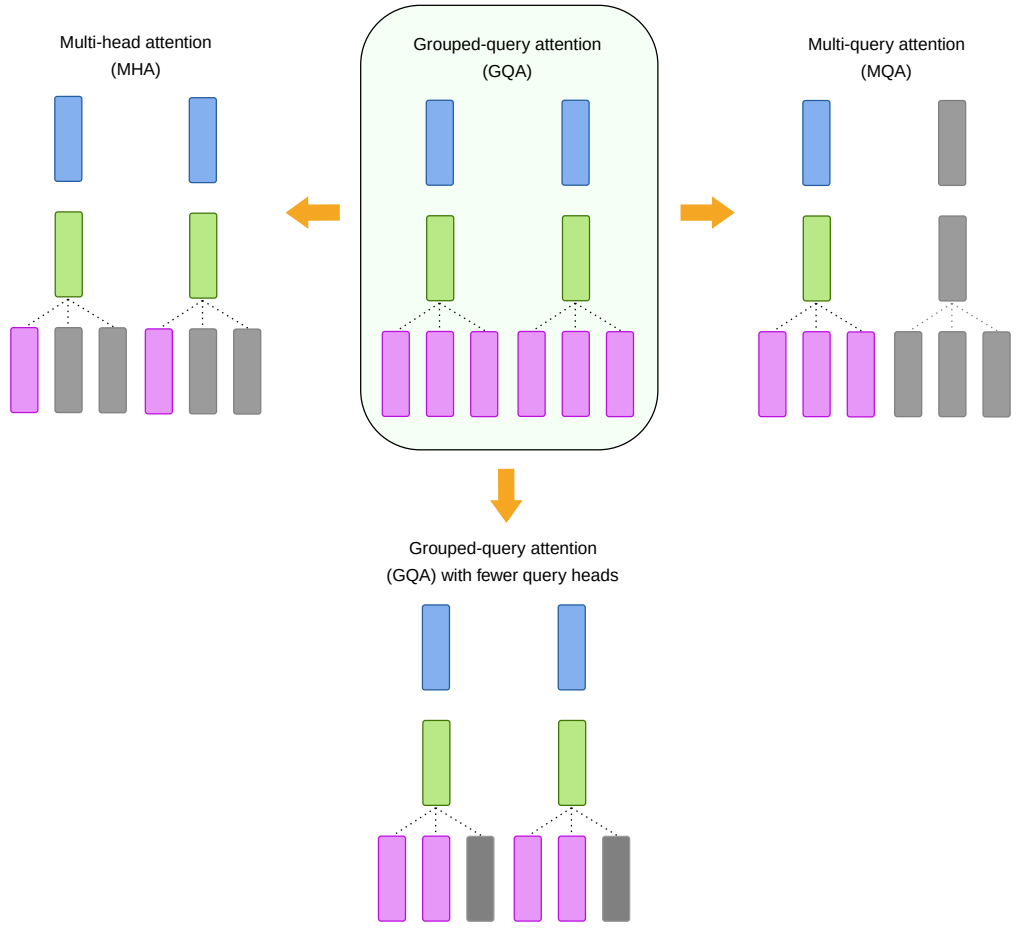


Figure 9: An example of how grouped-query attention (GQA) can be masked to emulate other forms of attention, such as multi-head or multi-query attention. The masked heads are shown in gray. Both MHA and MQA can be considered special cases of GQA. GQA with multiple groups and a single query per group yields MHA, while a single group with more than one query yields MQA. Note that GQA can also be reduced to fewer query heads while preserving the same number of groups.

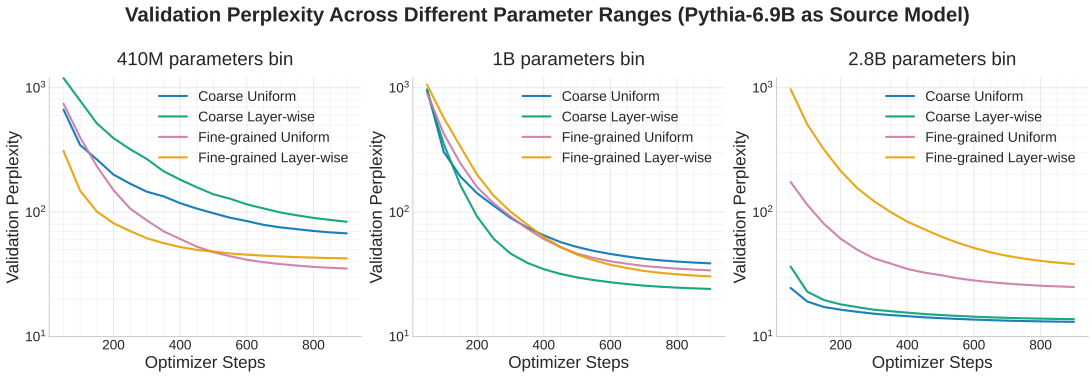


Figure 10: Validation perplexity of the best models from each search space found via evolutionary search. All models are initialized with Pythia-6.9B weights and trained for 2 billion tokens. Within each bin, the models' parameter counts fall within a ±5% range of that bin's target size.

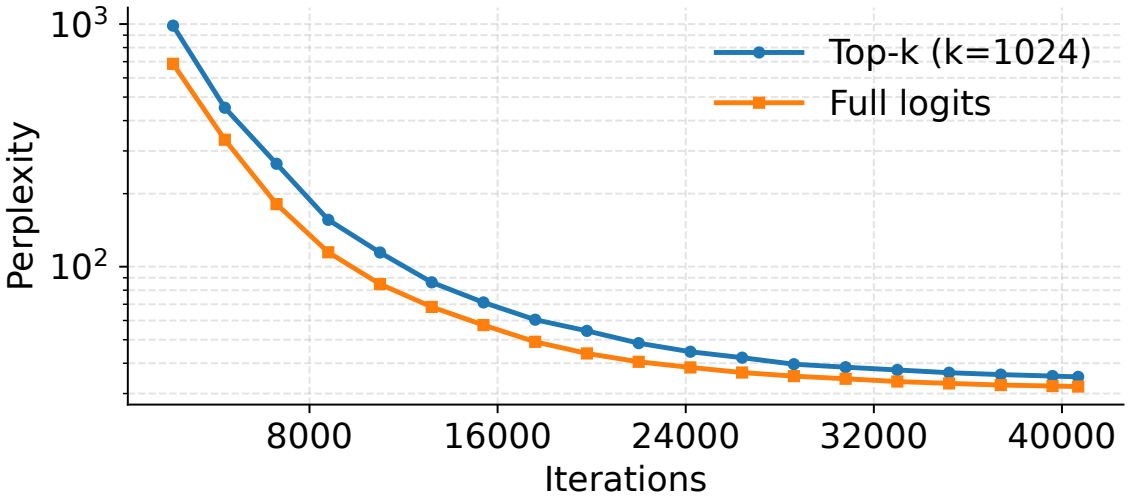


Figure 11: Full vs. top-k logit distillation.

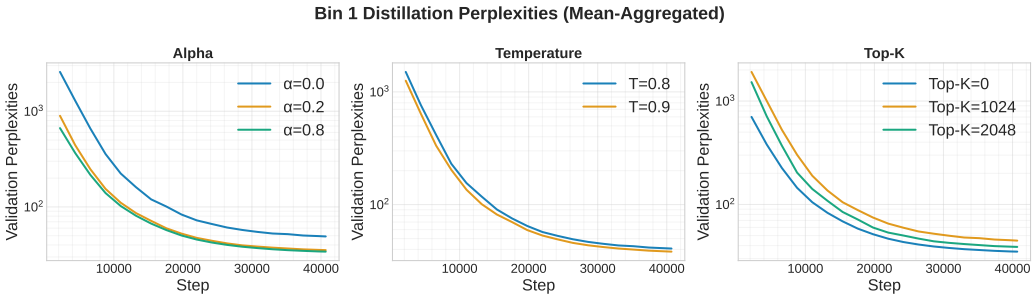


Figure 12: Bin 1 - distillation hyperparameter importance

3. T: the temperature parameter, which controls the smoothing of the teacher and student logit distributions.

4. $\mathcal{K}$: the number of top logits (Top-$\mathcal{K}$) used when computing the KL-divergence. A smaller $\mathcal{K}$ corresponds to a simpler distribution, while a larger $\mathcal{K}$ yields a more informative distribution. The maximum $\mathcal{K}$ equals the full number of teacher logits.

We now perform a grid-sweep over different choices of α, β, T and $\mathcal{K}$. We define the set of choices for α as $[0.2, 0.8, 0]$, the corresponding choices for β, which corresponds to $1 - \alpha$ as $[0.8, 0.2, 1]$, the choices for temperature T as $[0.8, 0.9]$ and the choices for $\mathcal{K}$ as $[1024, 2048, num_teacher_logits]$. Figures 12-14 present the perplexity curves aggregated for different hyperparameter values. In general, we observe that using only the distillation loss, i.e., setting $\alpha = 0$, is not recommended. Furthermore, a higher temperature and using the full logit distribution (Top-K = 0) perform best on average. In Table 18, we also present the importance of each of the hyperparameter choices and find that the most important one is the value of α, followed by $\mathcal{K}$ and finally the temperature T. Below, we present additional experimental results. Figure 15 shows the effect of using different teachers for knowledge distillation, Figure 16 shows the evolutionary search trajectory for different parameter bins, with the best perplexity marked in red. Table 20 presents the result of distilled models on downstream tasks. Table 22 provides the results on an extended set of common sense reasoning based downstream tasks.

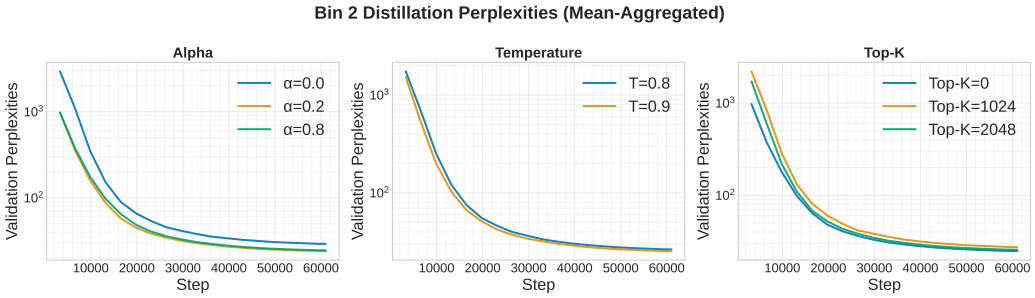


Figure 13: Bin 2 - distillation hyperparameter importance

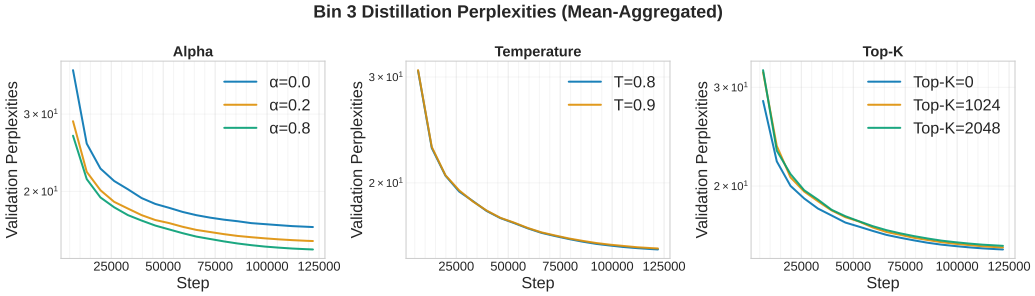


Figure 14: Bin 3 - distillation hyperparameter importance

hyperparameter	importance (mean)	importance (std)
α	0.986442	0.011966
$\mathcal{K}$	0.005968	0.000469
T	0.005000	0.000551

Table 18: Hyperparameter Importance

Figures 17–19 summarize the training behavior of the best models across different settings. The results highlight how architectures extracted from different search spaces (Figure 17), weight initialization strategies (Figure 18), and the use of distillation (Figure 19) affect convergence and final performance.

F Details on Importance Scoring

Importance scoring aims at defining scores for each transformer dimension, neuron or architecture parameter based on activation or weight magnitude. In our case, for a sub-network, the corresponding importance score serves as the proxy to sub-network quality or performance metrics like perplexity. The higher the importance score of a sub-network, the better its quality.

We adopt the dimension-wise importance scoring proposed by Muralidharan et al. [2024], which uses the activation of a component as proxy for its importance. Given a batch as input $\boldsymbol{X} \in \mathbb{R}^{B \times T \times d_{model}}$ after applying the embedding layer $\boldsymbol{W}^{emb}$ we compute the following scores for each component, where B corresponds to the batch dimension and T corresponds to the sequence length dimension, and *abs* corresponds to the absolute value function:

- For a neuron $i \in \{1, ..., U\}$ in a FFN layer l, we compute its importance by: $F^{(i)}_{FFN_l} = {}^1/_B \sum_B \left({}^1/_T \sum_T \mathbf{X} \boldsymbol{W}^l_1[:, i] \right)$ where $\mathbf{W}^l_1[:, i]$ corresponds to all weights of neuron i in layer l.

Base Model	Initialization	#Params	COPA	OpenBookQA	Lambada-OpenAI	Winogrande	Social IQA	MMLU-cont.	MMLU	CommonsenseQA	PIQA	ARC-challenge	ARC-easy	HellaSwag	BoolQ	Avg-acc	PPL-Nemotron-cc
Pythia-6.9B	Random Init	389M	59.00	29.20	18.51	51.14	36.59	25.82	26.43	19.41	63.87	24.74	46.80	31.28	58.17	37.77	26.20
	Supernet Init	389M	61.00	30.00	24.02	51.54	38.23	26.21	26.35	18.84	65.34	24.57	51.38	33.11	60.73	**39.33**	23.66
Pythia-12B	Random Init	407M	57.00	27.80	14.87	50.67	35.93	25.32	26.33	20.48	61.64	23.97	42.47	29.76	52.50	36.06	29.57
	Supernet Init	407M	63.00	27.40	18.37	52.09	37.10	25.90	25.99	21.21	62.73	23.55	46.42	30.91	52.78	37.50	27.33
Pythia-410M	Random Init	405M	62.00	29.60	19.54	50.67	36.49	25.72	25.54	20.15	64.14	24.57	47.35	32.70	61.01	38.42	25.29
Pythia-6.9B	Random Init	1.04B	64.00	29.20	23.36	51.54	38.59	26.63	26.62	21.05	65.78	27.13	51.80	36.83	60.36	40.22	21.84
	Supernet Init	1.04B	66.00	34.60	38.52	51.46	41.04	28.98	26.09	19.81	69.26	30.12	63.51	45.20	53.98	**43.74**	17.77
Pythia-12B	Random Init	1.04B	63.00	28.40	23.33	50.51	37.92	27.05	26.14	19.57	66.76	26.36	53.74	36.65	51.77	39.32	21.21
	Supernet Init	1.04B	64.00	31.20	27.56	51.77	38.48	27.36	26.19	19.82	66.54	26.45	53.96	36.42	61.65	40.88	20.77
Pythia-1B	Random Init	1.01B	64.00	30.20	25.67	52.41	38.95	26.93	25.20	20.97	66.00	28.24	56.14	38.23	60.83	41.06	20.11
Pythia-6.9B	Random Init	2.91B	61.00	30.60	26.49	52.10	39.00	27.60	26.39	19.82	67.74	28.33	57.83	41.12	59.05	41.31	13.75
	Supernet Init	2.91B	66.00	34.60	50.16	56.91	41.71	30.54	26.45	20.80	72.69	33.87	67.09	53.40	62.05	**47.41**	10.99
Pythia-12B	Random Init	2.91B	67.00	31.80	27.32	50.36	38.18	27.40	25.25	21.21	67.85	27.65	57.79	40.58	60.15	41.73	13.26
	Supernet Init	2.91B	69.00	33.20	41.76	51.46	40.79	29.29	26.20	21.21	70.57	28.67	61.99	45.98	58.04	44.47	11.71
Pythia-2.8B	Random Init	2.78B	68.00	30.4	24.51	53.03	39.50	27.18	25.74	20.47	67.68	25.34	46.67	39.11	59.54	40.55	14.54

Table 19: Evaluation of Pythia models across multiple benchmarks. Reported numbers are metrics as defined in Section 4.1.4 (%).

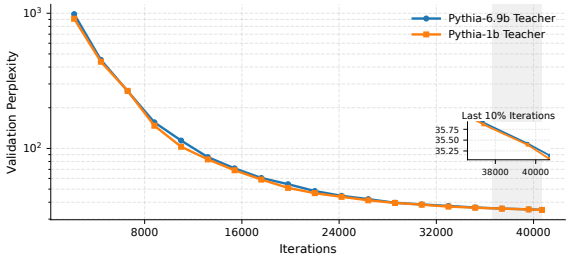


Figure 15: Teacher size vs. student performance with a 2B-token budget. A Pythia-1B teacher achieves validation perplexity 35.06, slightly better than Pythia-6.9B (35.15).

- Similarly for each neuron $i \in \{1, ..., d_{model}\}$ in the embedding layer we compute $F_{\text{emb}}^{(i)} = {}^{1}/_{B} \sum_{B} ({}^{1}/_{T} \sum_{T} (\text{Norm}(\mathbf{X}[:, :, i])))$. Specifically we perform mean absolute aggregation over output of every (Layer or RMS) Norm layer as

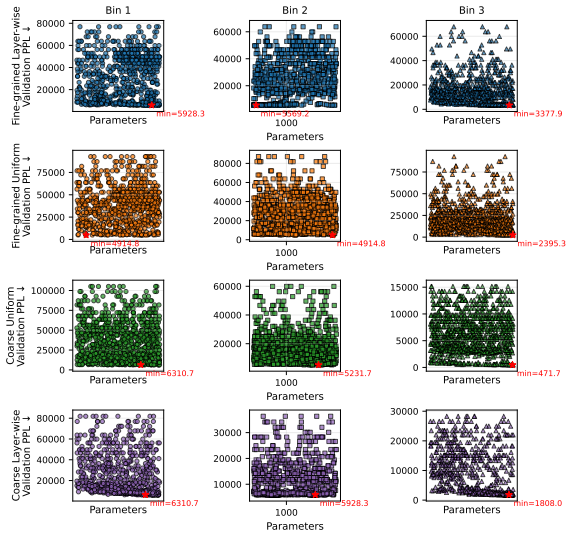


Figure 16: Evolutionary search on *coarse uniform*, *coarse layer-wise*, *fine-grained uniform* and *fine-grained layer-wise* search spaces for Pythia-6.9B. Minimum perplexity for each bin marked in red

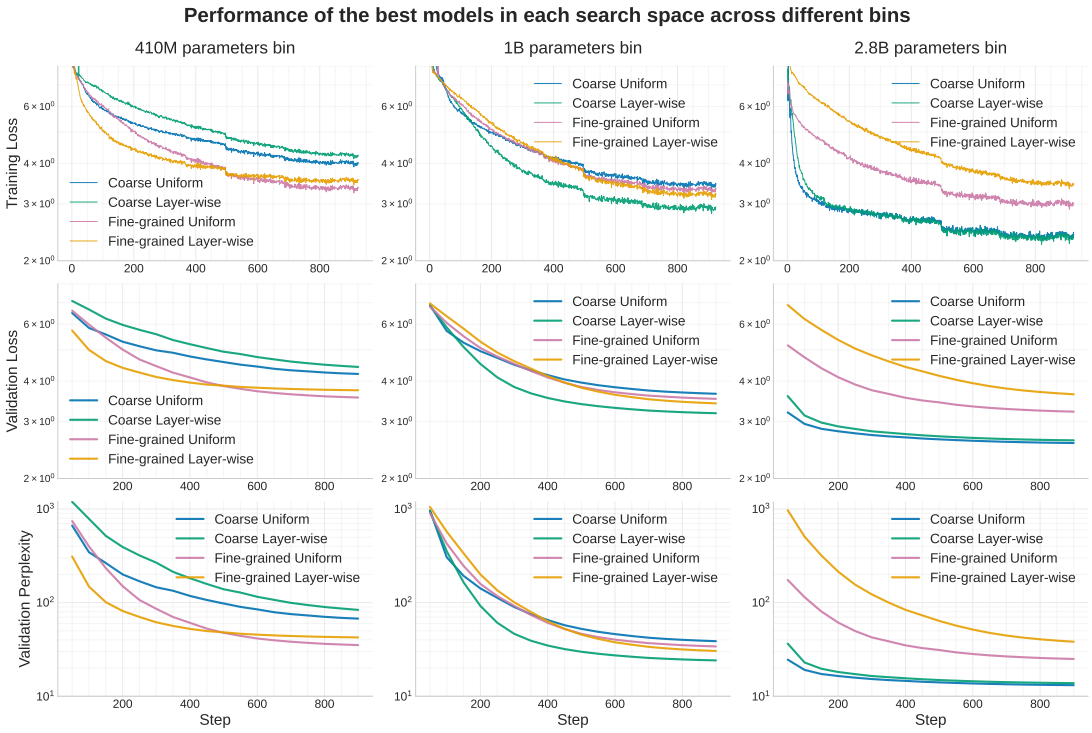


Figure 17: Training curves of the best models from each search space extracted from Pythia-6.9b (trained for 2 billion tokens)

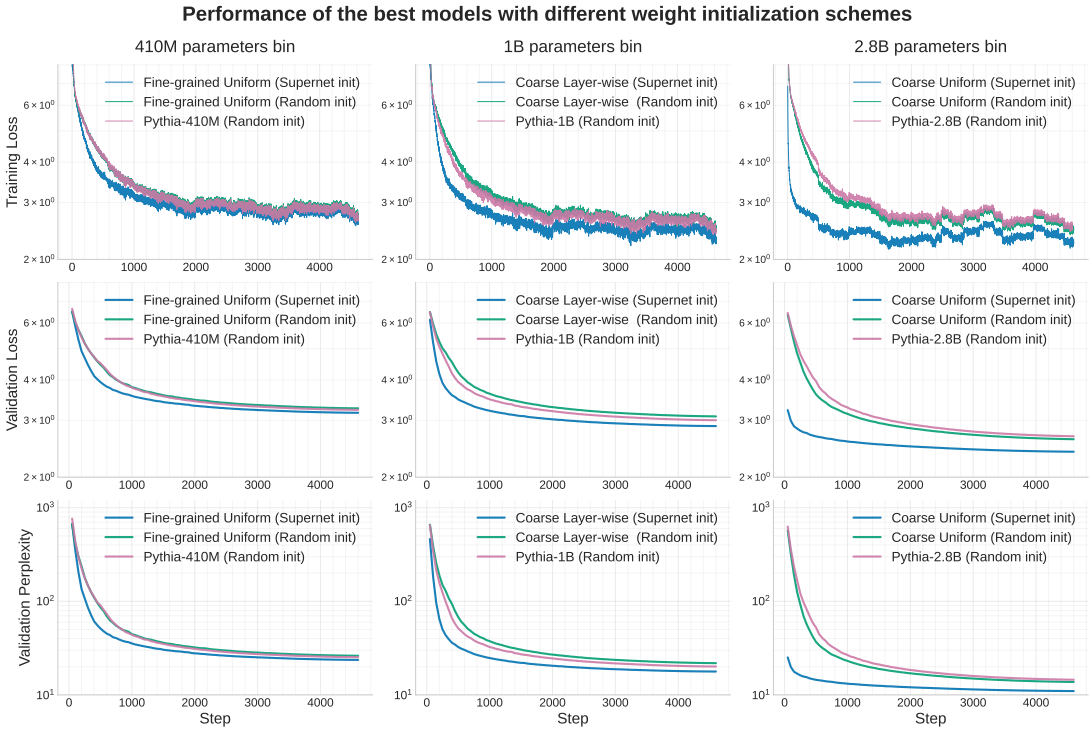


Figure 18: Training curves of the best models found in each bin, initialized with supernet weights as well as random weights. A Pythia model of comparable size is also trained with random initialization in each bin as a baseline. The models are trained with 10 billion tokens.

Initialization	#Params	COPA	OpenBookQA	Lambada-OpenAI	Winogrande	Social IQA	MMLU-cont.	MMLU	CommonsenseQA	PIQA	ARC-challenge	ARC-easy	HellaSwag	BoolQ	Avg-acc	PPL-Nemotron-cc
from-supernet	389M	61.00	30.00	24.02	51.54	38.23	26.21	26.35	18.84	65.34	24.57	51.38	33.11	60.73	39.33	23.66
from-supernet-distill	389M	66.00	30.60	23.95	49.57	37.41	26.31	25.62	19.57	65.56	25.34	49.66	33.92	54.31	39.06	18.59
from-supernet	1.04B	66.00	34.60	38.52	51.46	41.04	28.98	26.09	19.81	69.26	30.12	63.51	45.20	53.98	43.74	17.77
from-supernet-distill	1.04B	66.00	33.00	37.92	54.22	40.17	29.06	25.94	21.46	70.02	28.33	62.92	47.57	53.06	43.82	14.20

Table 20: Evaluation of sub-networks extracted from Pythia-6.9b for `bin-1` and `bin-2`. Reported numbers are metrics as defined in Section 4.1.4 (%). We compare training with the cross entropy loss (*from-supernet*) to training with knowledge distillation (*from-supernet-distill*) loss.

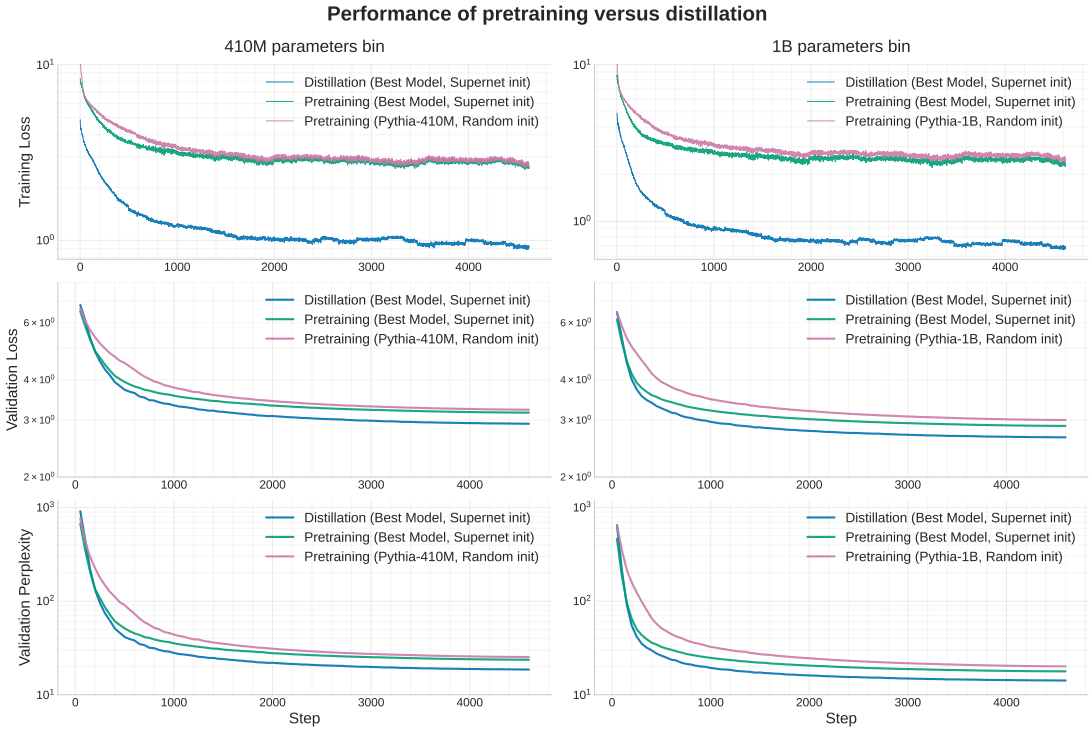


Figure 19: Training curves of the best models in bins 1 and 2, obtained through distillation and pretraining. In both cases, the models are initialized with weights from the supernet (Pythia-6.9B). For comparison, a Pythia model of similar size is trained from random initialization in both bins, serving as a baseline. All models are trained on 10 billion tokens.

- For causal attention layers we compute the importance of head $h \in \{1, ..., H\}$ of heads as :

$$F_{\text{MHA}}^{(h)} = {}^1/_B \sum_B \left({}^1/_T \sum_T \left\| \text{Attn}(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h) \right\|_2 \right)$$

- For a block $l \in \{1, ..., L\}$ consisting of a MHA and a FFN layer with RMS or layer normalization in between, we compute the score: $\text{F}_{block}^{(l)} = 1 - {}^1/_B \sum_B \left({}^1/_T \sum_T \left(\frac{\boldsymbol{X}_l^T \boldsymbol{X}_{l+1}}{\|\boldsymbol{X}_l\|_2 \|\boldsymbol{X}_{l+1}\|_2} \right) \right)$ where $\boldsymbol{X}_l$ is the input to block l and $\boldsymbol{X}_{l+1}$ the output.

Given, the score for each unit (layer, head or neuron), and a subnetwork configuration (for example: $e = 64$, $d = 128$, $l = 2$, $h = 4$), we compute the importance score corresponding the sub-network, by simply aggregating the normalized importance scores corresponding to the selected neurons, layers or heads. Similarly, for the weight space we define neuron, layer and head level importance scores by simply focussing on the magnitude of weight or neurons corresponding to every transformer dimension [Han et al., 2015].

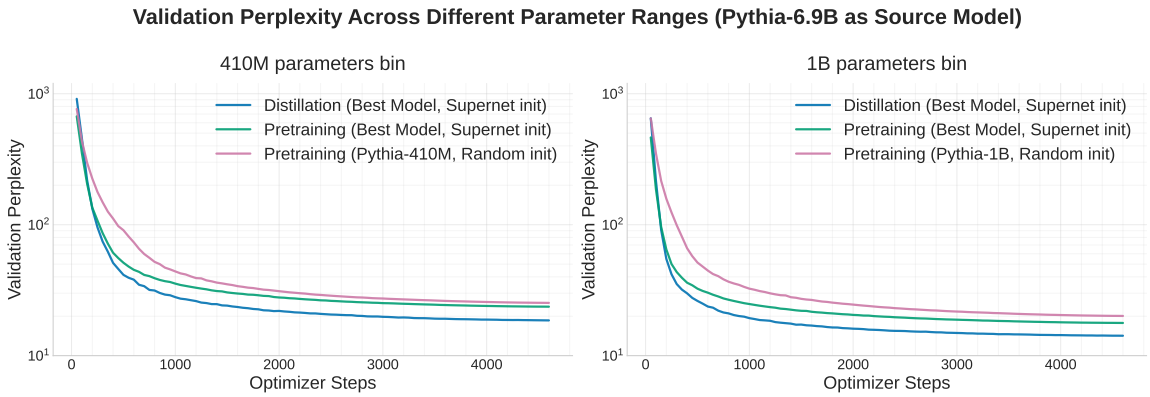


Figure 20: **Distillation**. Comparison of validation perplexity for models trained with distillation loss versus cross-entropy loss. All sub-networks are extracted from Pythia-6.9B as the base model and trained for 10B tokens.

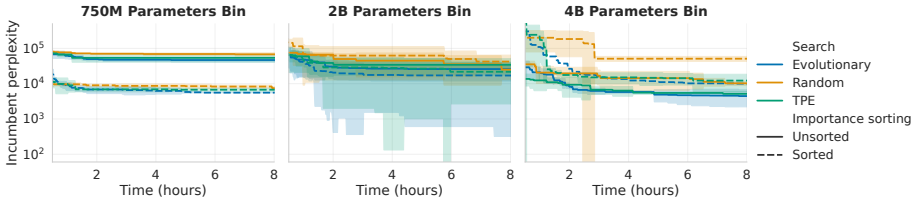


Figure 21: **Comparison of search methods on Qwen3**. The perplexities of the incumbent sub-network architecture for different search methods with and without importance sorting in the base checkpoint. The mean and standard deviation of three trials is visualized.

G Whittle APIs

An overview of the Whittle library is shown in Figure 7. In addition to the core functionalities of our framework described in Section 2, we provide an API to compute various importance metrics across different sub-network dimensions.

`set_sub_network()`. Given a pretrained decoder-only LLM from `litgpt`, we first convert it into a `whittle` model to enable flexible sub-network extraction. To evaluate a sub-network using the `whittle` model, we dynamically activate only structured components of the LLM associated with that sub-network using the `set_sub_network()` API (Listing 3). It allows the user to explicitly set architectural parameters of the sub-network, such as embedding dimension, intermediate size, number of heads, layers, query groups, and head size, as well as indices for sampled neurons, layers, and heads. Importantly, it allows to vary the number of heads, head size, intermediate size, and query groups across layers. This function is a core utility in `whittle`, supporting downstream procedures such as search, pretraining, and distillation.

`convert_subnet_to_litgpt_model()`. The `convert_subnet_to_litgpt_model()` function (Listing 2, Appendix G) transforms a selected sub-network configuration into a standalone GPT model within the `litgpt` framework. Given a super-network and a dictionary specifying architectural configurations (e.g., embedding dimension and number of heads), this utility extracts the corresponding sub-network and instantiates it as an independent GPT model. The resulting model can then be employed for downstream tasks such as pretraining, fine-tuning, or distillation.

`pretrain()`. The `pretrain()` function (Listing 1, Appendix G) enables pretraining of a sub-network initialized from a checkpointed GPT model. Given the model weights, a configuration file describing the sub-network architecture, and a target dataset, this utility restores the model and resumes training from the specified state.

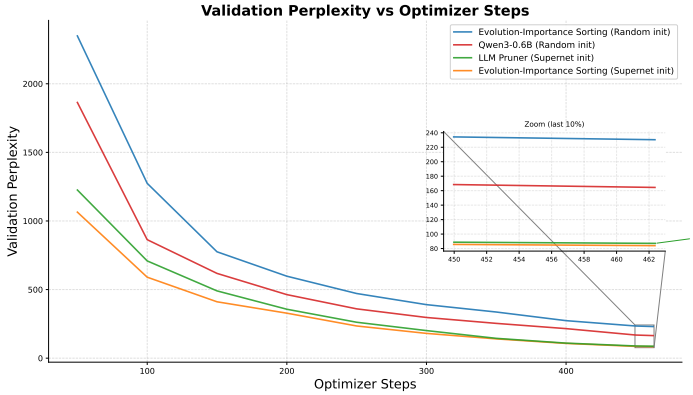


Figure 22: **Pretraining** Validation perplexity of the best sub-networks in the smallest bin for Qwen3. Qwen3-0.6B is the reference model. *Evolution-Importance Sorting* represents the best architecture that was found by searching the base model checkpoint which was sorted based on Minitron importance. *Random init* indicates that the model weights were initialized randomly before training, and Supernet init indicates that the sub-network was instantiated with the weights of the base model (Qwen3-8B with weights sorted by Minitron importance)

`distill()`. The `distill()` function (Listing 4, Appendix G) supports knowledge distillation from a larger teacher model into a sub-network extracted from a checkpoint. Given a teacher model, sub-network configuration, and a target dataset, this utility trains the sub-network under the supervision of a specified teacher (e.g., `EleutherAI/pythia-12b`). Different distillation objectives (e.g., forward KL divergence) and constraints such as top-k token selection are supported.

`compute_importance_score()`. The `compute_importance_score()` function (Listing 5) assigns an importance score to a given sub-network, where higher scores indicate higher estimated quality. Importance scores are computed independently for each architectural component (e.g., layers, heads, neurons), normalized across available choices using a softmax, and aggregated by summation. This computation is performed once at the beginning of the search procedure, after which evaluating the importance of candidate sub-networks becomes inexpensive compared to full metrics such as *perplexity*.

`evaluate_subnet()`. We provide an api from `whittle` [6] to lm-eval-harness[7] which is a popular library for evaluation of language models on common sense reasoning, math and coding benchmarks. The `evaluate_subnet()` provides functionality to set a desired subnetwork and then evaluate it on different tasks supported by lm-eval-harness. We present the api design of the evaluation functionality in Listing 6. We present the details of our API design for setting subnetwork in Listing 3, pretrain in Listing 1, convert to litgpt in Listing 2, distillation in Listing 4, importance metric computation in Listing 5 and the evaluation api in Listing 6.

H Search Space Sizes

The sizes of the search spaces for evolutionary search, especially for fine-grained (both uniform and layer-wise) can grow exponentially. In Table 21, we show the maximum number of configurations per search space for both of our base Pythia models (6.9B and 12B) and for Qwen3-8B in the Coarse Uniform search space. The search spaces and the equations to calculate their sizes are described below:

[6] `https://anonymous.4open.science/r/whittle-icml-AEB1/whittle/eval/whittle_llms.py`
[7] `https://github.com/EleutherAI/lm-evaluation-harness`

Listing 1: API for pretraining.

```
def pretrain(
     model_name: str, # Name of the model to load. E.g., EleutherAI/pythia-410m
     model_config: Optional[Config] = None, # Model configuration, overrides model_name
     config_path : Optional[str] = None, # Path to yaml file with model configuration,
    overrides model_config
     out_dir: Path = Path("out/pretrain"), # Path to save checkpoints to
     precision: Literal["bf16-true", "bf16-mixed", "32-true", None] = None,
     resume: Union[bool, Literal["auto"], Path] = False, # If true, resumes from the
    latest available checkpoint
     data: Optional[DataModule] = None, # Dataset to use to train
     train: TrainArgs = TrainArgs( # Training hyperparameters
         save_interval=1000,
         log_interval=1,
         global_batch_size=512,
         micro_batch_size=4,
         max_tokens=int(3e12),  # 3 trillion
         max_norm=1.0,
         min_lr=4e-5,
         lr_warmup_steps=2000,
         tie_embeddings=False,
     ),
     eval: EvalArgs = EvalArgs(interval=1000, max_iters=100), # Evaluation hyper-
    parameters
     optimizer: Union[str, Dict] = "AdamW", # Optimizer and its configuration
     devices: Union[int, str] = "auto", # CUDA or CPU
     num_nodes: int = 1, # Number of nodes for distributed training
     tokenizer_dir: Optional[Path] = None, # Path to tokenizer (optional)
     logger_name: Literal["wandb", "tensorboard", "csv", "mlflow"] = "tensorboard", #
    Logger to use
     seed: int = 42, # Seed for reproducibility
     init_from: str = "random", # Path to the checkpoint to load, or "random" to randomly
    initialize
     use_flex: bool = False, # Set to True if the sub-network has layer-wise configuration
) -> LitGPT
```

Listing 2: API for converting a subnet into a LitGPT model.

```
def convert_subnet_to_litgpt_model(
    supernet_name: str = "EleutherAI/pythia-12b",    # litgpt model to extract SLM from
    subnet_config: dict = {                          # subnet configuration
        sub_network_n_embd: 4,
        sub_network_intermediate_size: 16,
        sub_network_num_heads: 4,
        sub_network_n_layers: 2,
        sub_network_head_size: 4,
    }
) -> LitGPT                                  # returns LitGPT model
```

Listing 3: API for activating a sub-network

```
def set_sub_network(
    sub_network_n_embd: int = 4, # Embedding dim
    sub_network_intermediate_size: int | list[int] = 42,  # MLP size
    sub_network_num_heads: int | list[int] = 4, # No. of Attention heads
    sub_network_n_layers: int = 4, # No. of Layers
    sub_network_query_groups: int | list[int] = 2, # No. of Query groups
    sub_network_head_size: int | list[int] = 4,  # Head size
    sampled_intermediate_indices: list[int] | list[list] = [4,8], # Sampled MLP neurons
    sampled_head_indices: list[int] | list[list] = [2,3], # Sampled heads
    sampled_query_group_indices: list[int] | list[list] = [0,1], # Sampled query groups
    sampled_head_size_indices: list[int] | list[list] = [2,3,8,12], # Sampled head sizes
    sampled_layer_indices: list[int] = [2,3,5,6], # Sampled layers
    sampled_embd_indices: list[int] = [0,1,2,3], # Sampled embedding neurons
)
```

Coarse Uniform. The *coarse uniform* search space is the simplest possible design of a transformer. A sub-network is specified by selecting the number of layers $l \in \{1, \ldots, L\}$, the embedding dimension $e \in \{1, \ldots, E\}$, the number of attention heads $h \in \{1, \ldots, H\}$, the number of query groups $g \in \{1, \ldots, G\}$, the head dimension $h_s \in \{1, \ldots, H_s\}$, and the MLP intermediate size $d \in \{1, \ldots, D\}$. Each dimension is chosen as a *left subset* of the corresponding weight matrix. For example, selecting h heads corresponds to taking the first h heads from the H heads in T. All transformer layers share the same configuration (h, g, h_s, d), yielding *uniform* sub-networks. The size of this search space is:

$$|\mathcal{S}_{\text{coarse}}| = L \cdot E \cdot H_s \cdot D \cdot \mathcal{A}, \tag{6}$$

where $\mathcal{A}$ is the number of valid combinations of query heads and groups:

$$\mathbf{A} = \sum_{h=1}^{H} \sum_{g=1}^{G} \Big(\mathbf{1}[\, g = h \,] \;+\; \mathbf{1}[\, g = 1 \,] \;+\; \mathbf{1}[\, 1 < g < h \;\wedge\; h \bmod g = 0 \,] \Big), \tag{7}$$

corresponding to MHA when $g = h$, MQA when $g = 1$, and GQA when $1 < g < h$ with $h \bmod g = 0$.

Coarse Layer-wise. The *coarse layer-wise* search space relaxes the uniformity constraint of the coarse space: instead of enforcing the same configuration across all layers, each layer can independently choose its number of heads, query groups, head dimension, and MLP intermediate size. The search space size is:

$$|\mathcal{S}_{\text{coarse-layerwise}}| = E \cdot (H_s \cdot D \cdot \mathcal{A})^L, \tag{8}$$

or equivalently, in terms of $|\mathcal{S}_{\text{coarse}}|$ (equation 6):

$$|\mathcal{S}_{\text{coarse-layerwise}}| = E \cdot \left(\frac{|\mathcal{S}_{\text{coarse}}|}{L \cdot E} \right)^L. \tag{9}$$

Thus, moving from coarse to coarse layer-wise expands the search space from linear in L to exponential in L. In both cases, sub-networks are formed by selecting the left-most subset of neurons in each dimension.

Fine-grained Uniform. The *fine-grained uniform* search space extends the coarse setting by allowing arbitrary (not necessarily left-aligned) subsets of neurons, while keeping the same subset

Listing 4: API for distilling a sub-network from a checkpoint.

```python
def distill(
    teacher_checkpoint_dir: Path, # Path to teacher model checkpoint directory
    student_dir: Path, # Path to initialize student model directory with sub-network configuration and (optional) model weights
    data: DataModule | None = None, # Dataset for distillation
    out_dir: Path = Path("out/distill"), # Path to save distilled checkpoints
    precision: Literal["bf16-true", "bf16-mixed", "32-true", None] = None, # Precision for training
    train: TrainArgs = TrainArgs( # Training hyperparameters
        save_interval=1000,
        log_interval=1,
        global_batch_size=512,
        micro_batch_size=4,
        max_tokens=int(5e8),
        max_norm=1.0,
        min_lr=4e-5,
        lr_warmup_steps=2000,
        tie_embeddings=False,
    ),
    distill: DistillArgs = DistillArgs( # Distillation-specific hyperparameters
        method="logits", # Distillation method (e.g., logits, hidden states)
        temperature=10,  # Softening factor for teacher logits
        alpha=0.3,       # Weight for student loss
        beta=0.7,        # Weight for distillation loss
        loss="forward_kld", # Loss function for distillation
        weight_scheme="other", # Weighting scheme for combining losses
    ),
    eval: EvalArgs = EvalArgs(interval=50, max_iters=100, initial_validation=True), # Evaluation config
    optimizer: str | dict = "AdamW", # Optimizer and configuration
    devices: int | str = "auto", # CUDA or CPU
    num_nodes: int = 1, # Number of nodes for distributed distillation
    tokenizer_dir: Path | None = None, # Path to tokenizer (optional)
    logger_name: Literal["wandb", "tensorboard", "csv"] = "csv", # Logger backend
    seed: int = 42, # Seed for reproducibility
    random_init_student: bool = False, # If True, randomly initialize student instead of loading
) -> LitGPT # Returns a distilled LitGPT student model
```

Listing 5: API for computing importance scores of subnet components.

```
def compute_importance_score(
    supernet_name: str = "EleutherAI/pythia-12b",    # base supernetwork
    subnet_config: dict = {                          # subnet configuration
        sub_network_n_embd: 4,
        sub_network_intermediate_size: 16,
        sub_network_num_heads: 4,
        sub_network_n_layers: 2,
        sub_network_head_size: 4,
    },
    layer_importance_type: str  = "block importance", # method for layer scoring
    head_importance_type: str   = "minitron",         # method for head scoring
    neuron_importance_type: str = "minitron",         # method for neuron scoring
) -> int:   # returns importance score for a sampled sub-network
```

Listing 6: API for evaluating a `whittle` sub-network.

```
def evaluate_subnet(
    supernet_name: str = "EleutherAI/pythia-12b",    # litgpt model to extract SLM from
    subnet_config: dict = {                          # subnet configuration
        sub_network_n_embd: 4,
        sub_network_intermediate_size: 16,
        sub_network_num_heads: 4,
        sub_network_n_layers: 2,
        sub_network_head_size: 4,
    },
    task="mmlu",                                      # task name
    task_specifications= {                           # task hyperparameters
        "metric" = "accuracy",
        num_shot = 5
    }
) -> dict                                            # returns dictionary of results
```

selection fixed across all layers. The sub-network can also be formed from an arbitrary subset of layers rather than being restricted to the first l layers. The resulting search space grows as:

$$|\mathcal{S}_{\text{fine}}| = \Theta\big(2^E \cdot 2^H \cdot 2^{H_s} \cdot 2^D \cdot 2^L\big), \tag{10}$$

which is exponential in the layer width (E, H, H_s, D) and depth L.

Fine-grained Layer-wise. The *fine-grained layer-wise* search space applies fine-grained sampling independently to each layer, yielding the most granular search space considered. Each layer can independently select its number of heads, query groups, head dimension, and MLP intermediate size, and the sub-network may include an arbitrary subset of layers. The search space size is:

$$|\mathcal{S}_{\text{fine-layerwise}}| = \Theta\big(2^E \cdot (2^H \cdot 2^{H_s} \cdot 2^D)^L \cdot 2^L\big), \tag{11}$$

which grows exponentially with both the width (E, H, H_s, D) and the depth L, making it the largest and most expressive search space among the variants considered.

Search Space	Pythia-6.9B	Pythia-12B	Qwen3-8B
Coarse Uniform	4.33×10^{12}	9.51×10^{12}	3.7×10^{12}
Coarse Layer-wise	1.65×10^{244}	2.33×10^{281}	-
Fine-grained Uniform	10^{14284}	10^{17841}	-
Fine-grained Layer-wise	1.58×10^{159984}	5.31×10^{224611}	-

Table 21: Search space sizes (N) for the selected architectures. We do not consider the layer-wise and fine-grained search spaces for Qwen3-8B.

For the Qwen3-8B model, the head/query-group dimension, $h \in [1, 32]$ and $q \in [1, 8]$ are not independent due to GQA. A valid configuration requires h to be divisible by q and at most 4 heads per group ($h/q \leq 4$).

I Reproducibility Statements

We have taken extensive measures to ensure that all results in this paper can be replicated and verified by the community.

- **Code and Repository**: We release all our code and scripts to reproduce our experiments at `https://github.com/whittle-org/whittle` and `https://github.com/whittle-org/whittle-paper`.
- **Datasets and Pretrained Models**: We evaluate on available benchmarks from lm-eval-harness (`https://github.com/EleutherAI/lm-evaluation-harness`) and use the publicly available Nemotron-CC dataset `https://research.nvidia.com/labs/adlr/Nemotron-CC/` for training. Furthermore we use the Pythia-model suite, which is open-source `https://github.com/EleutherAI/pythia`.
- **Compute Resources**: All our search experiments were run on one L40 GPU per parameter bin and base model. All our pretraining runs for `bin-1` and `bin-2` were run on 8 L40 GPUs and `bin-3` was run on 4 H200 GPUs. All our distillation experiments were run on 4 H200 GPUs. We use cuda version 11.8.
- **Evaluation and Artifacts**: Upon acceptance of the paper we will publicly release model checkpoints for all our experiments.

J Scaling Behavior of Subnetwork Extraction Under Larger Budgets

To assess how the cost savings from subnetwork extraction scale with substantially larger pretraining budgets, we trained the best model from Bin 2 and Bin 3 for 100 billion tokens and compared it against a Pythia-1B and Pythia-2.8B model, respectively, trained for the same number of tokens from random initialization. We summarize our key findings below.

- **Sustained FLOP Savings at the 100B-Token Scale**. Our extracted subnetwork for Bin-3 achieves the same validation performance while requiring 1.26× fewer FLOPs (a reduction of approximately 21%). Although this reduction is smaller than the 5.16× savings observed at the 10B-token scale, it nevertheless demonstrates that subnetwork extraction continues to provide meaningful computational benefits even when the training budget is increased by an order of magnitude. This indicates that the method is not confined to low-budget regimes and remains competitive at significantly larger compute settings.

Base Model	Initialization	#Params	COPA	OpenBookQA	Lambada-OpenAI	Winogrande	Social IQA	MMLU-cont.	MMLU	CommonsenseQA	PIQA	ARC-challenge	ARC-easy	HellaSwag	BoolQ	Avg-acc
Pythia-6.9B	Supernet Init	1.04B	**74.00**	**38.40**	**45.604**	**56.98**	41.81	**32.54**	**27.04**	**20.88**	**75.03**	**38.57**	**71.76**	**60.05**	**61.19**	**49.53**
Pythia-1B	Random Init	1.01B	71.00	35.40	36.13	53.35	**41.91**	26.93	25.20	19.74	73.50	35.92	69.36	55.69	52.75	45.91
Pythia-6.9B	Supernet Init	2.91B	**76.00**	37.40	**53.23**	58.01	**42.02**	**34.45**	**38.04**	**43.41**	**77.69**	41.38	73.57	64.01	**63.49**	**54.05**
Pythia-2.8B	Random Init	2.78B	71.00	**40.40**	43.39	**58.64**	**42.02**	27.18	25.74	21.37	76.50	**42.75**	**74.24**	**64.85**	60.58	49.23

Table 22: Evaluation of Pythia models trained for 100B tokens across multiple benchmarks. Reported numbers are metrics as defined in Section 4.1.4 (%).

- **Improved Final Validation Perplexity**. In addition to being more compute-efficient, the extracted model also attains a lower final validation perplexity. The baseline Pythia-2.8B reaches a perplexity of 11.397, while our model achieves 11.204.

- **Strong Downstream Performance Advantages**. The extracted model outperforms the Pythia models trained from scratch across downstream evaluations. In particular, on MMLU-cont, our model achieves gains of up to 12% over the strongest Pythia baseline (see Table 22).

In Figures 23-25, we present the trajectories for validation perplexity, validation loss and train loss, respectively, for Bin 2 and Bin 3 architectures and the Pythia-based models trained from scratch for 100B tokens.

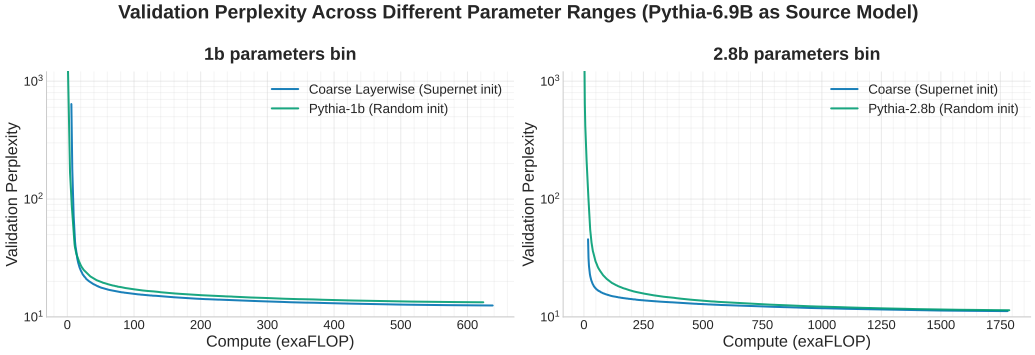


Figure 23: Bin 2 and Bin 3 validation perplexity for 100B token budget

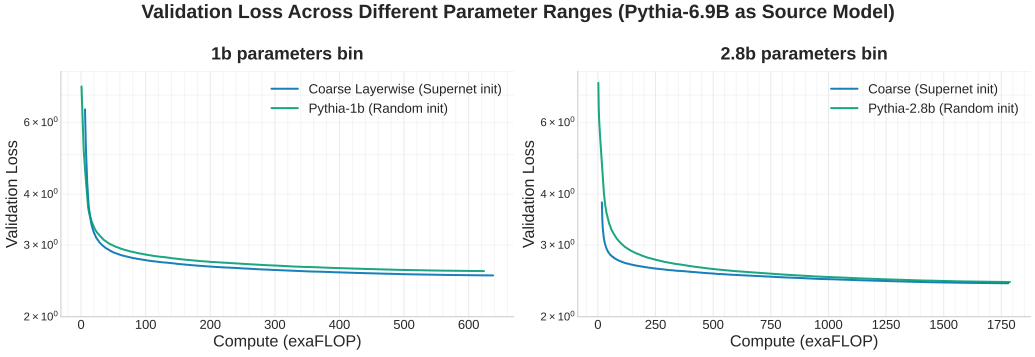


Figure 24: Bin 2 and Bin 3 validation loss for 100B token budget

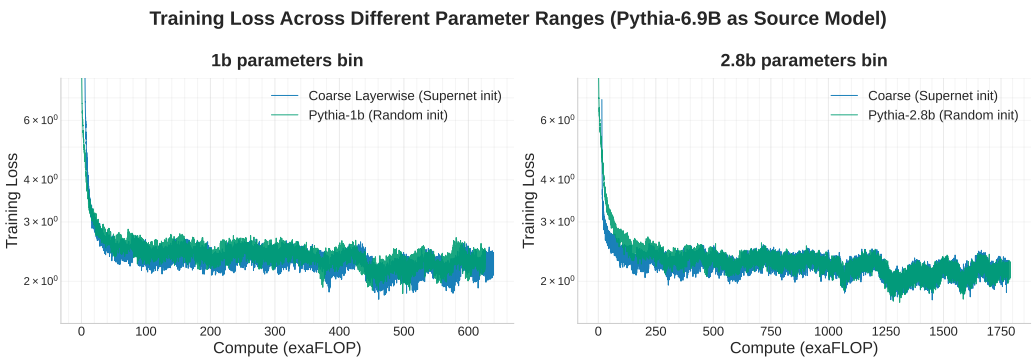


Figure 25: Bin 2 and Bin 3 train loss for 100B token budget

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The abstract and introduction describe `whittle` as a library for sub-network extraction, Minitron-style importance sorting, and pretraining/distillation pipelines. These claims are supported by Sections 2–4, including the 5.16× and 1.26× FLOP reductions on Pythia-6.9B/12B and the comparison of three search methods on Qwen3-8B.

   (b) Did you describe the limitations of your work? [Yes] The conclusion (Section 5) notes that extracted sub-networks inherit the limitations of their teacher models, and identifies scaling-law analysis and teacher-choice studies (e.g., multilingual vs. monolingual teachers) as open questions for future work.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The conclusion (Section 5) discusses both positive impacts (lowered barrier to entry, reuse of existing models, improved energy efficiency) and the fact that biases or other failure modes of the teacher model are inherited by the extracted student.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? (see `https://2022.automl.cc/ethics-accessibility/`) [Yes]

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources, etc.)? [Yes] All compared sub-networks within a given experiment use the same dataset (*Nemotron-CC* for pretraining, `wikitext` for search-time perplexity), the same token budgets (2B/10B/100B), and the same downstream evaluation suite via `lm-eval-harness`.

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning details and results, etc.)? [Yes] Search spaces are described in Section 3.2 and summarized in Table 1; full search-space sizes are reported in Appendix H; search and training hyperparameters are described in Sections 4.1.1–4.1.3 and Appendix B; downstream-evaluation shot counts and metrics are listed in Section 4.1.4.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] Search-method comparisons on Qwen3-8B (Figure 3) are run over three independent trials. Pretraining and distillation runs are conducted with a single seed per configuration owing to compute cost.

   (d) Did you report the uncertainty of your results (e.g., the standard error across random seeds or splits)? [Yes] Mean and standard deviation across the three search trials are visualized in Figure 3.

   (e) Did you report the statistical significance of your results? [No] We do not perform formal significance tests. We instead report mean and standard deviation across trials (where applicable) and consistent trends across three parameter bins and three base-model families.

   (f) Did you use enough repetitions, datasets, and/or benchmarks to support your claims? [Yes] We evaluate across three parameter bins, three base models (Pythia-6.9B, Pythia-12B, Qwen3-8B), four search spaces, three search methods, and eight downstream benchmarks (`copa`, `lambada_openai`, `winogrande`, `MMLU`, `piqa`, `arc-easy`, `arc-challenge`, `hellaswag`).

   (g) Did you compare performance over time and describe how you selected the maximum runtime? [Yes] Validation-perplexity-vs-token curves are shown in Figure 2. Pretraining

budgets (2B/10B/100B tokens) are specified in Section 4.1.2 and Appendix J; the 8-hour search budget for the Qwen3 search-method comparison is specified in Section 4.

(h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The Reproducibility Statements appendix reports the hardware used: a single L40 GPU per parameter bin and base model for search; 8×L40 for `bin-1`/`bin-2` pretraining; 4×H200 for `bin-3` pretraining and for all distillation runs; CUDA 11.8. Compute totals are also broken down in Appendix C.2 (Tables 13, 14, 15).

(i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We ablate over (i) four search-space granularities (Section 4.1.1), (ii) three search methods — random, evolutionary, and TPE (Section 4, Qwen3 experiments), (iii) Minitron-style importance sorting vs. unsorted base checkpoints (Figure 3), and (iv) supernet vs. random initialization at matched FLOP budgets (Figure 2, Table 2).

3. With respect to the code used to obtain your results…

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all dependencies (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation instructions, and execution commands (either in the supplemental material or as a URL)? [Yes] Code, installation instructions, and execution commands are publicly released at `https://github.com/whittle-org/whittle`; details are summarized in the Reproducibility Statements appendix.

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] The released repository includes example scripts and tutorial-style usage of the sub-network extraction, search, and pretraining APIs (also described in Appendix G).

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] `whittle` is an actively maintained open-source library; the API overview is provided in Appendix G, and the repository contains README, installation instructions, and usage examples.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] Reported downstream-evaluation numbers are given in Table 2 (and Table 20 for distillation). Model checkpoints for all experiments will be released publicly upon acceptance (Reproducibility Statements appendix).

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [No] The released repository at `https://github.com/whittle-org/whittle` contains the scripts used to produce the raw results, but not the figures/tables used in this paper

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes] We cite `litgpt` [AI, 2023], Pythia [Biderman et al., 2023], Qwen3 [Yang et al., 2025], Nemotron-CC [Su et al., 2025], `wikitext` [Merity et al., 2017], LLM-Pruner [Ma et al., 2023], and Minitron-style importance scoring [Muralidharan et al., 2024]. `lm-eval-harness` is referenced via its public repository.

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] We use only publicly released pretraining datasets and standard evaluation benchmarks that do not require individual consent.

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not curate any new datasets. The pretraining

corpus (Nemotron-CC) and evaluation benchmarks used are publicly released by their respective authors with their own data-handling documentation.

5. If you created/released new assets (e.g., code, data, models)…

   (a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] The license is included alongside the released code at `https://github.com/whittle-org/whittle`.

   (b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] The `whittle` library is released at `https://github.com/whittle-org/whittle`; trained model checkpoints will be released upon acceptance (Reproducibility Statements appendix).

6. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowdsourcing or human-subjects research was conducted.

   (b) Did you describe any potential participant risks, with links to institutional review board (IRB) approvals, if applicable? [N/A] No crowdsourcing or human-subjects research was conducted.

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] No crowdsourcing or human-subjects research was conducted.

7. If you included theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] The paper presents an empirical study; no theoretical results are claimed.

   (b) Did you include complete proofs of all theoretical results? [N/A] The paper presents an empirical study; no theoretical results are claimed.