LoopPrune: Evolutionary Module Selection with Iterative Execution for Efficient Large Language Model Compression

Shang Wang
Independent Researcher
email.shangwang@gmail.com

Abstract

Structured pruning offers hardware-agnostic acceleration for Large Language Models (LLMs) but typically degrades performance by strictly adhering to the original static sequential execution graph. In this paper, we challenge this rigidity with **LoopPrune**, an evolutionary framework that transforms model compression into an **architectural re-discovery** process. By treating pre-trained layers as a library of reusable primitives, LoopPrune optimizes a dynamic execution path where modules can be skipped, executed once, or iterated via controlled loops. This approach effectively **decouples effective model depth from parameter count**, allowing compressed models to maintain high reasoning capacity. To navigate the vast combinatorial search space, we introduce a hybrid evolutionary strategy enhanced by a **Multi-Tier Population Initialization** mechanism, which stratifies the search start-point using cross-model elite transfer, structural priors, and Wanda score importance estimation. Extensive experiments on diverse LLM families demonstrate that LoopPrune sets a new state-of-the-art across all sparsity levels. Notably, LoopPrune identifies efficient iterative patterns that allow compressed models to **match or even surpass the zero-shot reasoning accuracy of their dense counterparts** on specific benchmarks, while maintaining near-lossless perplexity at high retention rates. Code is available at https://github.com/ra225/LoopPrune.

1 Introduction

Large Language Models (LLMs) excel in natural language tasks, but their unprecedented scale poses significant challenges for deployment on resource-constrained hardware (Brown et al., 2020; Touvron et al., 2023). Consequently, model compression—particularly pruning—has emerged as a critical field of study to alleviate prohibitive computational costs.

Existing pruning paradigms typically fall into two categories: unstructured pruning (Frantar & Alistarh, 2023; Sun et al., 2024) and structured pruning (Fan et al., 2020; Ashkboos et al., 2024). Despite their successes, these methods share a fundamental limitation: they treat neural architectures as *static execution graphs*. In this paradigm, each selected module is executed exactly once sequentially. This rigid one-to-one mapping between memory presence and execution frequency creates a severe bottleneck.

We challenge this rigidity with **LoopPrune**, rooted in **Depth-Parameter Decoupling**. Structured pruning suffers from coupled degradation: removing a layer deletes both stored knowledge (parameters) and a non-linear reasoning step (depth). Traditional methods treat this coupling as an absolute physical constraint, causing unavoidable performance collapse at low retention rates. LoopPrune overcomes this by transforming compression into **architectural re-discovery**.

By selectively applying iterative execution (loops) to high-utility pre-trained primitives, we compensate for lost reasoning depth without introducing new parameters. This effectively decouples the

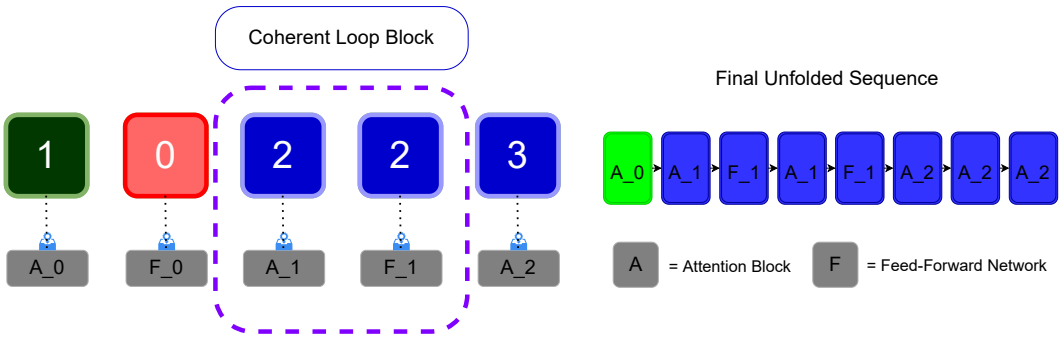


Figure 1: Schematic diagram of the chromosome decoding mechanism in LoopPrune. The figure illustrates the complete mapping from a discrete gene sequence (e.g., $c = [1, 0, 2, 2, 3]$) to a dynamic execution path. Pre-trained modules (Attention block A and FFN block F) are treated as reusable primitives. Gene values dictate the execution mode: 0 for pruning/skipping, 1 for standard single execution, and ≥ 2 for looped iterations. Notably, adjacent genes with identical values ≥ 2 are grouped into a "Coherent Loop Block" (e.g., the $[A_1, F_1]$ block repeated twice). This unfolding process demonstrates how LoopPrune decouples effective model depth from the physical parameter count.

model's effective inference depth from its physical memory footprint. Crucially, LoopPrune discovers non-sequential execution paths directly on off-the-shelf dense models to maximize parameter efficiency.

To navigate this expanded search space, LoopPrune employs an Evolutionary Algorithm (EA) enhanced by a **Multi-Tier Population Initialization** strategy. Our empirical evaluation on diverse LLM families (LLaMA2, LLaMA3.1, Qwen2.5, Qwen3, Qwen3.5) demonstrates that LoopPrune sets a new state-of-the-art across all sparsity levels. On downstream tasks like ARC-e, OBQA, and WinoGrande, our configurations even **surpass the original dense model**.

Our core contributions are summarized as follows:

- **Loop-based Execution Paradigm**: We redefine LLM pruning as execution path optimization, enabling modules to be skipped, executed once, or iterated.
- **Superior Reasoning Efficiency**: Iterative execution enables LoopPrune to achieve state-of-the-art perplexity and frequently **outperform dense models on zero-shot benchmarks**.
- **Decoupled Depth-Width Search**: Loop multiplicities discover architectures where effective depth is preserved or enhanced under parameter constraints.
- **Hybrid Optimization**: A stratified initialization strategy and hybrid search (GA + Local Search) yields substantial improvements over vanilla evolutionary methods.

2 Related Work

Structured Pruning. Model pruning includes unstructured and structured approaches. Unstructured methods like Magnitude (Han et al., 2015), SparseGPT (Frantar & Alistarh, 2023), and Wanda (Sun et al., 2024) achieve high sparsity by removing individual weights, but produce irregular matrices requiring specialized hardware for acceleration. Conversely, structured pruning removes coarse-grained blocks (e.g., attention heads or rows/columns), offering hardware-agnostic speedups. Among structured methods, LLM-Pruner (Ma et al., 2023) uses dependency graphs, SliceGPT (Ashkboos et al., 2024) employs matrix decomposition, ShortGPT (Men et al., 2025) removes layers with low Block Influence, and RPGE (Li et al., 2025) estimates gradients from outdated sub-models via reweighting. Recently, RoPT (Lu et al., 2026) demonstrated that directly pruning final layers followed by partial-layer fine-tuning yields strong performance and accelerates training recovery. Despite these advances, most structured methods treat models as static sequences, causing unavoidable degradation at high compression ratios.

Neural Architecture Search (NAS). NAS aims to automate the discovery of optimal network topologies (Real et al., 2019; Liu et al., 2018). To mitigate the prohibitive computational cost of training individual candidates, supernet-based NAS (or One-Shot NAS) methods (Pham et al., 2018; Cai et al., 2020; Guo et al., 2020; Wang, 2025a) propose training a single over-parameterized "supernet" where

all candidate sub-networks share weights. In the context of post-training compression, a pre-trained Large Language Model can be naturally viewed as a high-quality supernet. Rather than training a supernet from scratch, compression becomes a process of efficiently sampling sub-structures directly from the pre-trained weights. Evolutionary algorithms have proven effective for this discrete optimization problem (Javaheripi et al., 2022; Wang et al., 2023b,a; Wang, 2025b). Our work extends this supernet philosophy by expanding the search space beyond simple module selection to include dynamic execution paths, effectively treating the dense layers as a library of reusable primitives.

3 Method

3.1 Problem Formulation and Rationale

To operationalize the motivation of Depth-Parameter Decoupling, we formulate the compression process as a constrained execution path optimization problem. The core analytical challenge is that the introduction of loop multiplicities transforms the search space from a simple binary selection (retained vs. pruned) into an exponentially large, non-differentiable, and highly non-convex discrete landscape. Standard gradient-based architecture search or magnitude-based scoring, which assume monotonic performance degradation, are fundamentally inadequate here because they cannot evaluate the *synergistic effect* of repeatedly executing coherent functional blocks.

Consider a transformer-based Large Language Model (LLM) with L layers, where each layer ℓ consists of an attention module A_ℓ and a feed-forward network F_ℓ. We denote the sequence of all pre-trained modules as $\mathcal{M} = [A_0, F_0, A_1, F_1, \ldots, A_{L-1}, F_{L-1}]$, resulting in a total of $2L$ candidate primitives.

Standard structured pruning treats optimization as a binary selection problem: *which modules to retain?* We generalize this formulation by asking: *which modules to keep, and with what repetition multiplicity?* This yields an execution path $\pi = [m_1, m_2, \ldots, m_T]$, where any module from $\mathcal{M}$ may appear zero times (pruned), once (standard execution), or multiple times (iterative looping).

Our objective is to identify an optimal execution path π^* that minimizes calibration loss while adhering to a strict parameter budget. Crucially, the parameter cost is calculated based on the number of *unique* modules retained in memory, not the total number of execution steps T. Formally, the optimization problem is:

$$\pi^* = \arg\min_{\pi \in \Pi} \mathcal{L}_{\text{calib}}(\pi) \quad \text{s.t.} \quad \frac{|\theta_\pi|}{|\theta_{\text{full}}|} \leq \eta_{\text{target}} \tag{1}$$

where Π represents the space of valid execution paths that respect the topological ordering of the original computational graph, $\mathcal{L}_{\text{calib}}$ is the language modeling loss on a calibration set, $|\theta_\pi|$ denotes the count of unique parameters required to support path π, and η_{target} is the target retention ratio.

3.2 Loop-Based Chromosome Encoding

To explore this combinatorial search space efficiently, we introduce a loop-based chromosome encoding $\mathbf{c} \in \{0, 1, \ldots, K\}^{2L}$, where K represents the maximum allowable loop multiplicity. Each gene c_i determines the execution mode of the i-th module in $\mathcal{M}$:

- $c_i = 0$: Prune. The module is removed from memory and execution.
- $c_i = 1$: Standard. The module is executed exactly once.
- $c_i = k$ ($k \geq 2$): Loop Anchor. The module participates in a loop block of multiplicity k.

The decoding process, detailed in Algorithm 1, translates this chromosome into an execution path. A schematic diagram of this decoding mechanism is shown in Figure 1. A key feature of our decoding logic is the handling of consecutive loop genes: adjacent genes with identical values strictly greater than one are aggregated into a single coherent loop block that is iterated as a unit. This allows the search to discover both fine-grained module repetitions and coarse-grained layer-block repetitions.

This encoding provides a decoupled search space: *effective depth* (total operations) is controlled by the sum of gene values, while *model size* (memory footprint) is controlled by the count of non-zero genes. To rigorously ground this design, Appendix F provides a formal theoretical analysis demonstrating

how such iterative loop patterns act as expressivity amplifiers to maintain reasoning capacity under strict parameter budgets.

3.3 Evolutionary Search Framework: Bridging Algorithm and LLM Architecture

Given that the search space size scales exponentially as $\mathcal{O}(K^{2L})$, rendering brute-force or differentiable methods intractable, we design a specialized Genetic Algorithm. Crucially, our algorithmic components are not ad-hoc heuristics; rather, they are rigorously derived from the intrinsic structural properties of transformer networks.

3.3.1 Multi-Tier Population Initialization

Because the performance landscape of loop-enabled architectures is highly sensitive, purely random initialization frequently collapses into degenerate execution paths. To navigate this, we initialize the population $\mathcal{P}$ using a stratified strategy that acts as a strong structural prior:

1. Elite Pool Transfer. We exploit the observation that optimal execution motifs are strongly tied to the internal weight distributions of specific LLM families. By transferring high-performing paths from prior searches, adjusted for the target parameter retention rate, we bypass the barren plateaus of the search space.

2. Structural Priors. We seed a portion of the population with patterns known to be effective in sparse LLMs, such as U-shaped retention and heavily parameterized feed-forward configurations.

3. Wanda Score-Weighted Sampling. Leveraging the pruning metric from Sun et al. (2024), we estimate the sensitivity of each module via the product of weight magnitudes and input feature norms. We aggregate these element-wise scores to derive a module-level importance proxy. Genes are then initialized with probabilities proportional to their normalized scores, prioritizing structures with higher sensitivity.

4. Constrained Random. The remaining individuals are initialized randomly, subject to the constraint that the input embeddings and final classification head are always preserved.

3.3.2 Fitness Evaluation

The fitness function $f(\mathbf{c})$ is defined as the perplexity of the decoded model on a calibration dataset $\mathcal{D}_{\text{cal}}$:

$$f(\mathbf{c}) = \exp\left(-\frac{1}{N}\sum_{x\in\mathcal{D}_{\text{cal}}}\log p_{\pi(\mathbf{c})}(x)\right) \tag{2}$$

where N denotes the total number of tokens across all samples in $\mathcal{D}_{\text{cal}}$. To enforce the parameter budget, we apply a hard constraint during the generation of offspring. If a child chromosome exceeds η_{target}, we iteratively prune the gene with the lowest Wanda score until the budget is satisfied.

3.3.3 Genetic Operations

We evolve the population using a specific set of operators tailored for execution path discovery. Instead of treating genes as isolated variables, our operators respect the block-wise semantic processing nature of LLMs:

Selection & Crossover. We utilize an elite truncation selection and multi-point crossover. By exchanging coherent segments of execution paths rather than individual genes, we preserve functional sub-structures that act jointly to process specific semantic phases.

Mutation Strategies. To balance exploration and exploitation while respecting the model's topology, our framework builds upon a baseline genetic algorithm by integrating specialized mutation operators:

1. **Gradual Mutation:** Serving as the foundational exploration mechanism, this operator alters a single gene to an adjacent value. This reflects the hypothesis that local continuity exists in the performance landscape of loop multiplicities, enabling smooth baseline search.
2. **Block-Aware Mutation:** Transformers process information through multi-layer functional blocks, meaning that mutating isolated genes can sometimes disrupt phase continuity. To

mitigate this, this operator mutates a contiguous span of genes simultaneously to the same value, facilitating the discovery and preservation of deep iterative blocks as holistic units.

3. **Performance-Aware Mutation:** To optimize search efficiency and prevent premature convergence, this operator dynamically scales the mutation rate based on an individual's relative fitness. It protects high-performing structures from destructive topological changes while aggressively mutating low-performing candidates to escape local minima.

3.4 Hybrid Optimization via Local Search

While genetic algorithms are effective at global exploration, they lack sample efficiency for fine-grained refinement. We augment the evolutionary process with a periodic local search phase. At fixed intervals during evolution, we select the best-performing individual and apply a hill-climbing procedure. For each gene position, we evaluate the fitness impact of incrementing or decrementing the execution count. Updates that improve fitness without violating the parameter constraint are accepted immediately. This hybrid approach significantly accelerates convergence by fine-tuning the loop multiplicities of promising topologies.

3.5 Lightweight Normalization Recalibration

A pruned and looped execution path inherently exposes the pre-trained modules to activation statistics that deviate from the original dense model. In a standard dense architecture, the residual stream accumulates representations sequentially layer by layer. Crucially, modifying the execution path via skipping or looping primarily alters the magnitude and variance of this accumulated signal rather than fundamentally rotating the underlying semantic feature space, given that the core semantic extractors remain intact. Consequently, while standard affine projection weights exhibit strong robustness to these distributional shifts, normalization layers are disproportionately sensitive because their explicit architectural role is to govern these exact variances.

Therefore, rather than resorting to computationally expensive LoRA (Hu et al., 2022) or full fine-tuning, we introduce **Normalization Recalibration**. Specifically, we freeze all heavy attention and feed-forward parameters, optimizing exclusively the affine parameters of the normalization layers present in the discovered execution path:

$$\min_{\gamma,\beta} \sum_{x \in \mathcal{D}_{\text{tune}}} \mathcal{L}_{\text{NLL}}(x; \pi^*, \theta_{\text{frozen}}, \gamma, \beta) \tag{3}$$

This targeted step effectively realigns the feature distributions across the re-ordered modules by directly correcting the amplitude deviations caused by the altered residual accumulations. As rigorously verified in our experiments (Section 4.4), this approach successfully bridges the distribution shift with a minimal parameter footprint and negligible computational overhead.

4 Experiments

4.1 Experimental Setup

Search Hyperparameters. Our evolutionary search uses a population size of 100 over 100 generations, utilizing top-20 selection and two-point crossover with $p_c = 1.0$. The base mutation rate is set to $p_m = 0.05$. Initialization employs 60% elite transfer, 15% structural priors, 20% Wanda-guided, and 5% random. Local Search triggers every 10 generations. Further details are provided in Appendix B.

Datasets and Metrics. Following SliceGPT (Ashkboos et al., 2024), we use WikiText-2 (Merity et al., 2017) for language modeling evaluation. Specifically, we calculate fitness using 15 samples and perform RMSNorm recalibration on 1,024 samples (sequence length 1,024). Final perplexity is reported on the full test set with a sequence length of 2,048. For downstream evaluation, we measure zero-shot accuracy across seven standard reasoning benchmarks: ARC-Challenge, ARC-Easy (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and BoolQ (Clark et al., 2019).

Baselines. We use five state-of-the-art structured pruning methods—LLM-Pruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024), ShortGPT (Men et al., 2025), RPGE (Reweighted Policy Gradient

Table 1: Performance comparison of LoopPrune against state-of-the-art baselines across diverse LLM architectures at various parameter retention ratios. We report WikiText-2 perplexity (PPL↓) with sequence length 2048. Bold indicates the best performance at each ratio. All methods are evaluated under the same test setting.

Retention (%)	Method	LLaMA2		Qwen2.5		LLaMA3.1	Qwen3	Qwen3.5
		7B	13B	7B	14B	8B	8B	9B
100	Dense	5.50	4.90	6.86	5.32	6.22	9.71	8.62
90	LLM-Pruner	8.87	8.52	9.76	9.94	9.54	14.21	11.83
	ShortGPT	8.15	6.82	9.68	9.15	7.95	13.24	10.11
	SliceGPT	7.41	7.31	9.29	8.68	8.42	12.56	10.51
	RPGE	6.99	6.67	8.67	8.37	8.12	11.95	10.12
	RoPT	6.84	6.45	8.42	7.54	7.61	11.83	9.97
	LoopPrune (ours)	**5.62**	**5.01**	**7.64**	**6.15**	**6.83**	**10.64**	**9.25**
80	LLM-Pruner	10.83	9.56	15.82	14.25	12.87	18.54	15.92
	ShortGPT	10.22	7.94	14.52	12.64	10.14	17.51	12.86
	SliceGPT	9.29	8.31	13.77	11.38	11.23	16.42	14.15
	RPGE	8.27	7.44	10.45	9.87	10.55	14.22	12.45
	RoPT	8.56	7.82	11.53	9.91	9.54	14.36	12.63
	LoopPrune (ours)	**6.71**	**5.64**	**8.71**	**7.11**	**7.39**	**12.19**	**10.02**
70	LLM-Pruner	17.03	15.98	39.32	29.17	20.15	38.62	28.45
	ShortGPT	14.85	10.15	28.41	18.47	13.48	30.15	16.53
	SliceGPT	12.73	11.67	25.66	16.25	15.31	27.81	19.34
	RPGE	16.41	12.35	34.97	32.51	22.45	45.67	38.74
	RoPT	14.23	12.91	27.84	17.65	17.26	29.54	21.45
	LoopPrune (ours)	**8.25**	**6.59**	**11.12**	**9.60**	**9.64**	**15.44**	**12.43**
50	LLM-Pruner	39.78	26.31	69.10	56.41	45.32	72.19	52.47
	ShortGPT	36.19	19.46	58.17	36.92	30.65	59.88	32.47
	SliceGPT	31.61	18.11	53.29	32.67	35.18	55.43	37.82
	RPGE	48.59	29.39	104.70	79.75	62.14	115.38	85.91
	RoPT	35.18	21.05	56.72	35.21	38.45	61.27	41.56
	LoopPrune (ours)	**13.84**	**10.03**	**25.04**	**16.94**	**22.98**	**32.06**	**23.24**

Estimator) (Li et al., 2025), and RoPT (Reverse-order Pruning and Partial-layer Tuning) (Lu et al., 2026)—as baselines. All results are obtained by running their official code.

Search cost. Experiments run on NVIDIA A100 80GB GPUs. For LLaMA2-7B, search takes 2.7 GPU hours and fine-tuning an additional 0.5 GPU hours.

4.2 Main Results

Table 1 reports the WikiText-2 perplexity of LoopPrune and baselines across diverse LLM families (LLaMA2/3.1, Qwen2.5/3/3.5) under 50%–90% parameter retention. The experimental results demonstrate that LoopPrune consistently maintains a leading position across all settings, effectively breaking the bottlenecks of traditional pruning through architectural re-discovery.

First, LoopPrune achieves near-lossless performance across various architectures under moderate compression. Notably, on LLaMA2-13B at 90% retention, it achieves a perplexity of 5.01, closely tracking the original dense model (4.90) and substantially outperforming the strongest baseline, RoPT (6.45).

Second, the advantages of our approach become particularly pronounced under extreme compression at 50% parameter retention. Traditional static pruning methods suffer from catastrophic performance degradation at this level due to the severe loss of non-linear reasoning depth. For instance, on Qwen3.5-9B, methods like RPGE deteriorate drastically to a perplexity of 85.91, while ShortGPT (32.47) and SliceGPT (37.82) also exhibit significant performance collapse. In stark contrast, LoopPrune successfully compensates for this lost depth via iterative execution, maintaining a highly competitive perplexity of 23.24.

Finally, LoopPrune significantly improves parameter efficiency, successfully shifting the Pareto frontier of model compression. For example, Qwen3-8B at only 70% retention (15.44) surpasses the performance of SliceGPT operating at a much higher 80% retention rate (16.42). This decoupling

Table 2: Zero-shot accuracy evaluation on Qwen2.5-14B and Qwen3.5-9B across multiple sparsity levels (70%–90% retention). Bold indicates the best performance at each ratio. Underlined values indicate where LoopPrune surpasses the original dense model.

Model	Method	Ratio (%)	Zero-shot Accuracy ↑							
			ARC-c	ARC-e	Wino	Hella	OBQA	PIQA	BoolQ	Avg
Qwen2.5 -14B	Dense	100	58.87	79.17	75.37	82.90	45.00	82.10	85.23	72.66
	RPGE	90	49.40	71.09	70.17	69.82	44.40	75.19	72.63	64.67
	RoPT	90	51.45	76.14	72.14	74.15	44.80	77.86	75.14	67.37
	LoopPrune (ours)	90	**54.95**	**81.61**	**74.11**	**78.68**	**45.20**	**80.96**	**77.80**	**70.47**
	RPGE	80	48.72	69.70	69.53	63.81	**44.60**	75.24	71.07	63.24
	RoPT	80	51.11	74.16	71.11	70.45	44.00	77.15	73.15	65.88
	LoopPrune (ours)	80	**53.75**	**79.29**	**73.72**	**77.41**	43.40	**80.63**	**75.57**	**69.11**
	RPGE	70	33.36	42.59	57.06	61.17	31.20	59.19	58.41	49.00
	RoPT	70	42.15	58.12	65.11	65.13	36.20	69.10	65.14	59.40
	LoopPrune (ours)	70	**52.56**	**75.04**	**73.32**	**76.15**	**42.40**	**80.25**	**73.33**	**67.58**
Qwen3.5 -9B	Dense	100	55.80	74.16	73.09	78.07	42.60	80.30	89.17	70.46
	RPGE	90	48.63	65.82	66.30	69.45	37.40	73.61	82.57	63.40
	RoPT	90	52.39	71.46	70.88	74.22	40.80	77.80	86.36	67.70
	LoopPrune (ours)	90	**54.61**	**73.86**	**74.19**	**76.84**	**42.20**	**79.54**	**88.13**	**69.91**
	RPGE	80	43.17	59.43	60.14	62.15	34.20	68.44	76.15	57.67
	RoPT	80	49.57	67.59	66.46	69.84	38.60	74.54	82.45	64.15
	LoopPrune (ours)	80	**52.39**	**70.96**	**69.69**	**73.42**	**40.80**	**77.15**	**85.44**	**67.12**
	RPGE	70	36.69	50.13	53.67	54.21	30.20	61.43	68.32	50.66
	RoPT	70	44.88	61.24	61.88	63.85	35.80	69.80	77.25	59.24
	LoopPrune (ours)	70	**49.15**	**66.84**	**65.90**	**68.55**	**38.80**	**74.16**	**81.53**	**63.56**

of effective depth from parameter count enables the compressed models to maintain formidable language modeling capabilities even under severe resource constraints.

4.3 Downstream Zero-Shot Reasoning

To evaluate how effectively LoopPrune preserves complex reasoning capabilities beyond language modeling perplexity, Table 2 presents the zero-shot accuracy of Qwen2.5-14B and Qwen3.5-9B across seven standard benchmarks under 70%–90% parameter retention (extended evaluations on the remaining model families, extreme 50% compression, and generative coding/math benchmarks are provided in Appendix A).

Surpassing Dense Counterparts via Iterative Execution. A remarkable finding in Table 2 is that LoopPrune not only outperforms static pruning baselines across all configurations, but also identifies efficient iterative patterns that allow compressed models to *surpass the accuracy of their uncompressed dense counterparts*. Specifically, on Qwen2.5-14B, LoopPrune exceeds the dense baseline on ARC-e at both 90% retention (81.61% vs. 79.17%) and 80% retention (79.29% vs. 79.17%), as well as on OBQA at 90% retention (45.20% vs. 45.00%). Similarly, on Qwen3.5-9B at 90% retention, LoopPrune outperforms the dense model on WinoGrande (74.19% vs. 73.09%) while maintaining near-lossless average accuracy (69.91% vs. 70.46%). This empirically validates our core hypothesis that static sequential execution is not strictly optimal, and selectively looping high-utility primitives can amplify reasoning capacity.

Robustness as Compression Tightens. As the parameter retention ratio decreases to 70%, static pruning methods suffer steep accuracy drops due to the permanent removal of non-linear reasoning steps. In contrast, LoopPrune's Depth-Parameter Decoupling compensates for lost physical layers through looped execution, outperforming the strongest baseline (RoPT) in average zero-shot accuracy by **+8.18%** on Qwen2.5-14B and **+4.32%** on Qwen3.5-9B at 70% retention.

4.4 Effectiveness of RMSNorm Fine-tuning

To rigorously verify the core assumption that RMSNorm-only fine-tuning is sufficient to bridge the execution-induced distribution shift, we compare it against SliceGPT's full LoRA-based fine-

Table 3: Fine-tuning comparison on LLaMA2-13B. LoopPrune uses RMSNorm-only fine-tuning, while SliceGPT uses its official default LoRA configuration (r=32, α=10). Our approach achieves better improvements with 100$\times$ fewer trainable parameters and up to 5.5$\times$ faster training.

Retention	Method	Perplexity		Improvement	Trainable Params		Time
		Before	After		Count	% of Total	
90%	LoopPrune	5.80	**5.01**	**13.6%**	363K	0.0031%	0.6h
	SliceGPT	8.27	7.31	11.6%	120.6M	0.89%	3.3h
80%	LoopPrune	6.76	**5.64**	**16.6%**	317K	0.0031%	0.6h
	SliceGPT	9.28	8.31	10.5%	116.0M	0.97%	3.1h
70%	LoopPrune	8.27	**6.59**	**20.3%**	282K	0.0031%	0.6h
	SliceGPT	14.51	11.67	19.6%	111.5M	1.08%	2.9h
50%	LoopPrune	17.27	**10.03**	**41.9%**	200K	0.0031%	0.6h
	SliceGPT	23.21	18.11	22.0%	102.3M	1.41%	2.7h

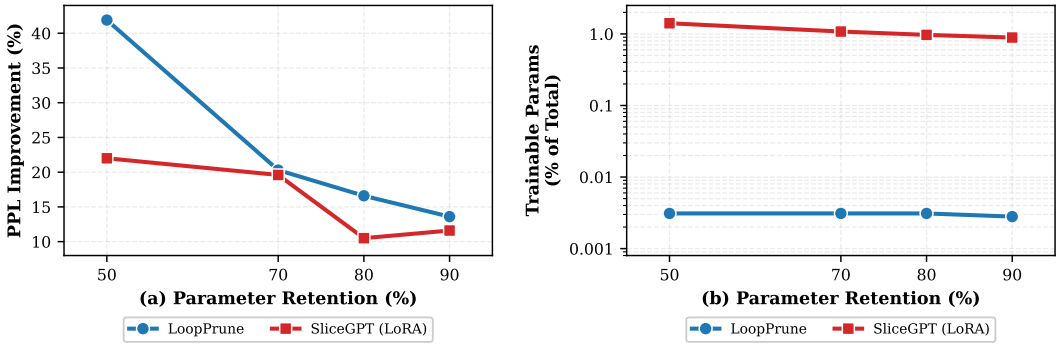


Figure 2: Fine-tuning efficiency comparison on LLaMA2-13B. **(a)** Perplexity improvement after fine-tuning. LoopPrune achieves consistently better improvements, especially at high compression (50% retention). **(b)** Trainable parameters as percentage of total model size (log scale). LoopPrune maintains constant $\sim$0.003% across all sparsity levels, while LoRA requires 100–600$\times$ more parameters and scales poorly with compression.

tuning (rank-32, α=10) across multiple sparsity levels on LLaMA2-13B. As detailed in Table 3, our targeted recalibration achieves superior performance improvements while training exponentially fewer parameters.

Key observations: (1) LoopPrune consistently achieves larger perplexity improvements, especially at higher compression ratios (41.9% vs 22.0% at 50% retention). (2) Our RMSNorm-only approach trains 100–600$\times$ fewer parameters than LoRA (200K–363K vs 102M–121M), yet delivers superior results. (3) Training time is reduced by 4.5–5.5$\times$ (0.6h vs 2.7–3.3h), with the speedup increasing at higher compression. (4) The trainable parameter percentage remains nearly constant across sparsity levels for LoopPrune ($\sim$0.003%), while LoRA's percentage increases with compression (0.89% to 1.41%), indicating our method scales better to aggressive pruning.

Figure 2 visualizes the efficiency advantage: LoopPrune achieves better improvements with orders of magnitude fewer trainable parameters. The gap widens at higher compression ratios, where LoRA requires increasingly more parameters relative to model size while our method maintains constant efficiency.

This empirical evidence rigorously validates our hypothesis that RMSNorm recalibration is not only theoretically justified but empirically sufficient for adapting to dynamic execution paths, demonstrating a decisive efficiency advantage over general-purpose adaptation methods like LoRA.

4.5 Experiments on Extra-Large Language Models

We further evaluate the scalability of LoopPrune on extra-large models (Qwen3.5-27B and LLaMA2-70B).

Table 4: Results of extra large language models. Perplexity (PPL↓) is reported on WikiText2 where SliceGPT is calibrated at sequence length 1024. Methods are evaluated under the same test setting.

Model	Dense	Pruning Ratio (%)							
		10		20		30		50	
		SliceGPT	Ours	SliceGPT	Ours	SliceGPT	Ours	SliceGPT	Ours
Qwen3.5-27B	6.86	7.81	**7.09**	8.89	**7.42**	11.02	**8.41**	22.91	**15.43**
LLaMA2-70B	3.32	3.74	**3.56**	4.63	**4.19**	5.59	**4.94**	9.32	**7.40**

Table 5: Inference efficiency comparison on LLaMA2-70B. We follow the standard benchmark protocol used in SliceGPT (Ashkboos et al., 2024): input length $L_{\text{in}} = 128$ tokens, output length $L_{\text{out}} = 128$ tokens, batch size 1, FP16 precision. Metrics are defined as follows: (1) **Mem(GB)** represents the peak GPU memory footprint during inference; (2) **E2E latency** is the end-to-end generation time computed as $T_{\text{E2E}} = T_{\text{prefill}} + T_{\text{decode}}$, where T_{prefill} is the time to process the input prompt and T_{decode} is the total time to generate L_{out} tokens autoregressively with KV cache; (3) **Tok/s** (throughput) is calculated as Tok/s $= L_{\text{out}}/T_{\text{decode}}$, representing the number of tokens generated per second during the decode phase; (4) **Speedup** is the ratio of E2E latency between the dense baseline and the pruned model, i.e., $S = T_{\text{E2E}}^{\text{dense}}/T_{\text{E2E}}^{\text{pruned}}$. Each result is averaged over 10 runs with 5 warmup iterations to ensure stable GPU frequency and cache states.

Method	Ratio(%)	Params(B)	Params(%)	Mem(GB)	E2E(ms)	Tok/s	Speedup
Dense	100	68.98	100.0	151.9	7316.8	17.92	1.00×
SliceGPT	90	70.78	102.6	160.2	7863.8	16.62	0.93×
Ours	90	**61.98**	**89.9**	**137.9**	**6304.7**	**20.81**	**1.16**×
SliceGPT	80	62.15	90.1	139.6	7207.8	18.12	1.02×
Ours	80	**55.08**	**79.9**	**123.5**	**5698.9**	**22.94**	**1.28**×
SliceGPT	70	53.64	77.8	122.7	6445.3	20.28	1.14×
Ours	70	**48.19**	**69.9**	**108.8**	**5111.4**	**25.73**	**1.43**×
SliceGPT	50	37.44	54.3	85.2	5970.5	21.79	1.22×
Ours	50	**34.35**	**49.8**	**77.5**	**4748.5**	**27.58**	**1.54**×

4.5.1 Accuracy under Large-Scale Pruning

Table 4 reports WikiText-2 perplexity results under different pruning ratios. LoopPrune consistently outperforms SliceGPT across all models and sparsity levels, with the gap becoming more pronounced at higher compression.

A key factor behind this difference is parameter efficiency. SliceGPT relies on additional orthogonal rotation matrices to reshape the parameter space, which significantly increases the effective parameter count. As a result, even under the same nominal sparsity, SliceGPT retains substantially more parameters than LoopPrune, and at 10% sparsity its total parameter count can even exceed that of the original dense model. In contrast, LoopPrune strictly enforces the parameter budget by counting only unique retained modules.

4.5.2 Inference Efficiency

We evaluate LoopPrune's scalability on extra-large models: Qwen3.5-27B and LLaMA2-70B. As shown in Table 5, LoopPrune consistently achieves lower end-to-end latency and higher decoding throughput than SliceGPT. At 50% retention, LoopPrune reaches a 1.54× speedup over the dense model, compared to 1.22× for SliceGPT.

These results indicate that LoopPrune translates parameter reduction more directly into inference speedup, while SliceGPT's auxiliary rotation parameters introduce additional memory and computation overhead (at 90% retention, this overhead even causes a net slowdown). Overall, LoopPrune achieves a superior trade-off between accuracy, parameter efficiency, and inference speed at extra-large model scales. Extended evaluations on memory scaling, KV-cache behavior, and decoding throughput across larger batch sizes are provided in Appendix C.

4.6 Ablations

4.6.1 Elite Pool Transfer

To validate the efficacy of our initialization strategy, we conducted an ablation study on LLaMA2-13B at a 70% parameter retention ratio (population size 50, 50 generations). We compared a baseline of random initialization against two elite transfer sources: (1) *Cross-Model Transfer* (from LLaMA2-7B at 70% retention) and (2) *Cross-Sparsity Transfer* (from LLaMA2-13B at 50% retention). Figure 3 illustrates the convergence trajectories.

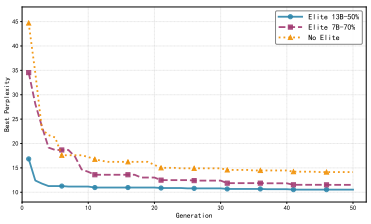


Figure 3: Convergence trajectories of the best-performing individuals on LLaMA2-13B at 70% sparsity using different elite initialization strategies.

Based on these results, we derive two key insights regarding the search efficiency: (1) Superior Convergence and Performance: Initializing with historical elites significantly accelerates the search process. Both transfer strategies converge to near-optimal solutions within the first 20 generations, whereas random initialization struggles to escape local minima. This is reflected in the final perplexity scores, where elite transfer yields substantial gains over the random baseline. (2) Intrinsic Structure Over Sparsity: Crucially, *Cross-Sparsity Transfer* (same model, different ratio) outperforms *Cross-Model Transfer* (different model, same ratio). This suggests that optimal execution paths are highly specific to the model family's internal weight distribution. Consequently, transferring structural knowledge from the same model—even at a different sparsity level—provides a higher-quality prior than matching the target sparsity using a different architecture.

4.6.2 Advanced Mutation and Hybrid Optimization

To isolate the contributions of our proposed optimization mechanisms, we conducted an ablation study on LLaMA2-13B at 50% sparsity. Starting from a standard baseline Genetic Algorithm (GA) which exclusively utilizes the Gradual Mutation operator, we incrementally integrated our specialized mechanisms: Block-aware mutation, Performance-aware (Adaptive) mutation, and periodic Local Search.

Table 6: Ablation study of search components on LLaMA2-13B at 50% sparsity. Each row represents the cumulative addition of a specific operator to the baseline Genetic Algorithm.

Configuration	Final PPL	Gain
Baseline GA	19.24	-
+ Block-aware mutation	18.73	+2.7%
+ Performance-aware mutation	18.37	+4.5%
+ Local search (every 10 gen)	**17.27**	**+10.2%**

As detailed in Table 6, each component yields a monotonic improvement in model fidelity. Most notably, the integration of Local Search delivers the most substantial gain (improving the baseline by 10.2%). This suggests that while advanced mutation strategies aid in global exploration, the hybrid approach of coupling evolution with fine-grained local exploitation is critical for refining the precise loop multiplicities required for high-performance compression (further ablations in Appendix D).

5 Conclusion

We introduced LoopPrune, a framework that redefines structured pruning as dynamic execution path optimization rather than static module removal. By treating pre-trained layers as reusable primitives, our evolutionary approach decouples effective depth from parameter count. Extensive experiments across multiple generations of LLM architectures, including the LLaMA (v2, v3.1) and Qwen (v2.5, v3, v3.5) families, demonstrate that LoopPrune consistently achieves state-of-the-art performance across all sparsity levels. It successfully maintains strong reasoning capabilities even at extreme 50% parameter retention, where traditional methods suffer from catastrophic collapse. Notably, our discovery that optimized iterative paths can occasionally surpass dense models on specific benchmarks fundamentally challenges the optimality of fixed sequential execution. Combined with lightweight RMSNorm recalibration, LoopPrune offers a practical pathway for efficient deployment and establishes a new paradigm for transforming foundation models into flexible, highly capable compact architectures.

References

Ashkboos, S., Croci, M. L., do Nascimento, M. G., Hoefler, T., and Hensman, J. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vXxardq6db.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision*, pp. 544–560. Springer, 2020.

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

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Javaheripi, M., de Rosa, G., Mukherjee, S., Shah, S., Religa, T., Teodoro Mendes, C. C., Bubeck, S., Koushanfar, F., and Dey, D. Litetransformersearch: Training-free neural architecture search for efficient language models. *Advances in Neural Information Processing Systems*, 35:24254–24267, 2022.

Li, Z., Liao, Y., Wu, B., Zhou, Y., Shi, X., Jiang, D., Li, Y., and ZHANG, W. Computation and memory-efficient model compression with gradient reweighting. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=qILtlupwrR.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

Lu, Y., Cheng, H., Fang, Y., Wang, Z., Wei, J., Xu, D., Xuan, Q., and Zhu, Z. Reassessing layer pruning in LLMs: New insights and methods. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=04Tfwy3LLC.

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

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Men, X., Xu, M., Zhang, Q., Yuan, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PxoFut3dWW.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wang, S. Elastic architecture search for efficient language models. In *2025 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6. IEEE, 2025a.

Wang, S. W-pca based gradient-free proxy for efficient search of lightweight language models. In *International Conference on Learning Representations*, volume 2025, pp. 42419–42440, 2025b.

Wang, S., Tang, H., and Ouyang, J. A neural architecture search method using auxiliary evaluation metric based on resnet architecture. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 687–690, 2023a.

Wang, S., Tang, H., and Ouyang, J. A transformer-based neural architecture search method. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 691–694, 2023b.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

A Additional Zero-Shot and Generative Reasoning Benchmarks

This section provides the extended downstream reasoning evaluations complementing Section 4.3, including zero-shot accuracy across the remaining five model architectures (LLaMA2-7B, LLaMA2-13B, Qwen2.5-7B, LLaMA3.1-8B, and Qwen3-8B), extreme 50% compression across all seven model families, as well as advanced generative reasoning benchmarks (HumanEval, MBPP, and MATH500).

A.1 Extended Zero-Shot Accuracy Benchmarks

We organize the extended zero-shot evaluations across seven standard benchmarks (ARC-Challenge, ARC-Easy, WinoGrande, HellaSwag, OpenBookQA, PIQA, and BoolQ) into three tables:

Performance on LLaMA2 and Qwen2.5 Architectures: Table 7 reports the zero-shot accuracy for LLaMA2-7B, LLaMA2-13B, and Qwen2.5-7B under moderate to high parameter retention ratios (70%–90%). LoopPrune consistently outperforms state-of-the-art structured pruning baselines, demonstrating that dynamic execution path optimization preserves core reasoning skills significantly better than static module removal.

Generalization to LLaMA3.1 and Qwen3: Table 8 evaluates LLaMA3.1-8B and Qwen3-8B across the 70%–90% retention range, confirming that LoopPrune establishes a new state-of-the-art across diverse foundation model families.

Robustness Under Extreme Compression (50%): Table 9 aggregates the evaluation of all seven model families under an extreme compression constraint of 50% parameter retention. While traditional baseline methods exhibit severe performance degradation due to the loss of physical reasoning depth, LoopPrune sustains robust zero-shot accuracy via Depth-Parameter Decoupling.

A.2 Advanced Generative Reasoning

To rigorously test generative and structural reasoning, which require strict algorithmic logic and precise edge-case handling, we evaluate the Qwen3.5-9B model on HumanEval, MBPP, and the MATH (500 subset) benchmarks.

As shown in Table 10, standard static pruning permanently severs the sequential reasoning chain, frequently causing catastrophic failures in these fragile tasks (e.g., at 50% retention, LLM-Pruner and RPGE collapse to $< 20\%$ on HumanEval). LoopPrune's iterative execution effectively sustains the structural integrity necessary for complex deductions, massively outperforming all baselines and preserving substantial reasoning capacity even under aggressive compression.

B Implementation Details

B.1 Decoding Strategy and End-to-End Evolutionary Pipeline

To facilitate a comprehensive understanding of the LoopPrune framework, we detail the core algorithmic procedures in this section.

Chromosome Decoding. Algorithm 1 specifically details the chromosome decoding strategy utilized during fitness evaluation and final model execution. This decoding mechanism is responsible for translating the discrete loop-based chromosome into a valid forward-pass execution path, properly handling the unrolling of coherent loop blocks.

End-to-End Pipeline. Algorithm 2 outlines the complete execution process of the LoopPrune framework. The pipeline consists of three main phases: (1) multi-tier initialization to jump-start the search, (2) the core evolutionary loop enhanced by periodic local search, and (3) final model extraction and lightweight recalibration.

B.2 Elite Transfer Mapping Function

When transferring an elite chromosome from a source model M_S (with $2L_S$ modules) to a target model M_T (with $2L_T$ modules), we apply a proportional index mapping function $\phi : \mathbb{Z}^{2L_S} \to \mathbb{Z}^{2L_T}$.

Table 7: Zero-shot accuracy evaluation on LLaMA2-7B, LLaMA2-13B, and Qwen2.5-7B across multiple sparsity levels (70%–90% retention). Bold indicates the best performance at each ratio.

Model	Method	Ratio (%)	Zero-shot Accuracy ↑							
			ARC-c	ARC-e	Wino	Hella	OBQA	PIQA	BoolQ	Avg
LLaMA2 -7B	Dense	100	46.25	74.62	69.06	76.01	44.20	79.11	77.68	66.70
	RPGE	90	**42.49**	61.32	63.61	63.21	37.40	71.71	63.61	57.62
	RoPT	90	41.81	64.98	64.48	65.34	39.20	73.18	64.89	59.13
	LoopPrune (ours)	90	41.55	**68.69**	**65.35**	**68.17**	**40.80**	**77.37**	**66.97**	**61.27**
	RPGE	80	34.73	58.25	58.48	57.78	**38.80**	70.89	60.58	54.22
	RoPT	80	35.58	61.11	59.83	60.42	37.60	72.47	61.59	55.51
	LoopPrune (ours)	80	**36.43**	**64.06**	**61.01**	**64.73**	38.20	**74.70**	**62.60**	**57.39**
	RPGE	70	30.20	42.00	52.41	47.00	32.20	64.25	54.68	46.11
	RoPT	70	31.83	48.32	53.12	54.45	34.20	68.44	55.41	49.40
	LoopPrune (ours)	70	**33.62**	**60.14**	**54.38**	**61.29**	**36.80**	**73.56**	**57.80**	**53.94**
LLaMA2 -13B	Dense	100	49.06	77.44	71.98	79.39	45.20	80.52	80.55	69.16
	RPGE	90	37.80	62.58	65.11	66.29	38.20	73.18	64.92	58.30
	RoPT	90	38.65	65.40	65.43	69.84	40.20	75.14	66.33	60.14
	LoopPrune (ours)	90	**40.10**	**69.36**	**66.14**	**73.73**	**42.00**	**77.37**	**68.44**	**62.45**
	RPGE	80	35.41	59.43	59.67	63.28	36.60	72.36	61.83	55.51
	RoPT	80	36.18	62.16	60.85	65.43	38.20	74.37	62.63	57.12
	LoopPrune (ours)	80	**37.20**	**65.36**	**62.27**	**69.91**	**39.60**	**76.22**	**63.88**	**59.21**
	RPGE	70	30.97	42.85	53.51	52.95	33.20	66.21	55.81	47.93
	RoPT	70	32.17	52.65	54.14	59.45	35.20	70.46	57.16	51.60
	LoopPrune (ours)	70	**34.30**	**61.36**	**55.49**	**66.09**	**37.60**	**75.08**	**58.99**	**55.56**
Qwen2.5 -7B	Dense	100	51.11	77.36	72.85	78.93	47.20	79.71	84.68	70.26
	RPGE	90	**49.15**	69.65	68.75	68.42	43.60	73.67	71.19	63.49
	RoPT	90	48.12	72.14	69.14	70.45	44.20	75.63	74.13	64.83
	LoopPrune (ours)	90	47.61	**76.52**	**70.64**	**74.15**	**45.00**	**78.13**	**77.80**	**67.12**
	RPGE	80	45.82	68.31	**70.32**	62.54	40.60	73.72	69.66	61.57
	RoPT	80	46.08	71.25	69.30	68.45	42.20	75.41	71.16	63.41
	LoopPrune (ours)	80	**48.04**	**74.45**	68.11	**73.00**	**43.80**	**77.86**	**74.07**	**65.62**
	RPGE	70	32.51	41.75	55.88	39.04	30.80	58.00	57.25	45.03
	RoPT	70	39.42	56.10	62.12	52.14	36.20	66.10	64.13	53.74
	LoopPrune (ours)	70	**47.01**	**71.55**	**69.93**	**73.11**	**41.60**	**76.17**	**71.87**	**64.46**

Specifically, the i-th gene in the target chromosome is assigned the value from the source chromosome at index:

$$j = \min\left(\left\lfloor i \times \frac{2L_S}{2L_T} \right\rfloor, 2L_S - 1\right) \tag{4}$$

This nearest-neighbor interpolation strictly scales the lengths of contiguous loop blocks, thereby preserving their structural integrity and front-middle-back execution patterns during the cross-model transfer process.

B.3 Adaptive Mutation Rate Formulation

The adaptive mutation strategy scales the base mutation rate p_m dynamically according to the individual's relative performance. Let $\rho = f(\mathbf{c})/\bar{f}$ denote the performance ratio, where $f(\mathbf{c})$ is the individual's perplexity and $\bar{f}$ is the population average perplexity. The adjusted mutation rate p'_m is defined as a piecewise function:

$$p'_m = \begin{cases} 0.5p_m, & \text{if } \rho < 0.9 \text{ (Elite performers)} \\ p_m, & \text{if } 0.9 \leq \rho \leq 1.1 \text{ (Average performers)} \\ 1.5p_m, & \text{if } \rho > 1.1 \text{ (Poor performers)} \end{cases} \tag{5}$$

Table 8: Zero-shot accuracy evaluation on LLaMA3.1-8B and Qwen3-8B across multiple sparsity levels (70%–90% retention). Bold indicates the best performance at each ratio.

Model	Method	Ratio (%)	Zero-shot Accuracy ↑							
			ARC-c	ARC-e	Wino	Hella	OBQA	PIQA	BoolQ	Avg
LLaMA3.1 -8B	Dense	100	53.41	81.14	73.32	78.92	44.80	81.23	82.08	70.70
	RPGE	90	48.38	68.60	69.22	66.90	40.40	**78.45**	74.92	63.84
	RoPT	90	47.10	71.21	68.11	69.37	41.20	75.68	76.39	64.15
	LoopPrune (ours)	90	**50.17**	**74.71**	**70.64**	**72.75**	**42.60**	77.64	**78.41**	**66.70**
	RPGE	80	43.77	65.74	65.35	60.58	38.00	70.62	73.27	59.62
	RoPT	80	45.65	**72.10**	66.93	64.12	39.40	72.69	71.16	61.72
	LoopPrune (ours)	80	**48.55**	71.51	**69.30**	**69.68**	**41.40**	**75.90**	**76.57**	**64.70**
	RPGE	70	39.76	54.17	62.12	53.03	36.80	66.21	66.67	54.11
	RoPT	70	41.89	58.33	63.85	57.03	35.20	68.55	69.05	56.27
	LoopPrune (ours)	70	**45.31**	**65.11**	**66.69**	**63.50**	**39.20**	**72.31**	**72.91**	**60.72**
Qwen3 -8B	Dense	100	56.40	80.93	67.88	74.93	41.40	77.75	86.61	69.41
	RPGE	90	49.15	67.97	63.77	63.37	38.40	72.63	79.88	62.17
	RoPT	90	50.60	70.62	64.56	**69.50**	37.60	71.33	78.13	63.19
	LoopPrune (ours)	90	**52.65**	**74.28**	**65.75**	69.00	**39.40**	**74.48**	**82.26**	**65.40**
	RPGE	80	45.31	61.20	61.56	60.71	35.60	67.95	73.67	58.00
	RoPT	80	47.44	64.98	62.83	57.30	36.80	69.86	76.18	59.34
	LoopPrune (ours)	80	**50.77**	**70.96**	**64.72**	**66.04**	**38.40**	**72.80**	**80.09**	**63.40**
	RPGE	70	43.17	57.37	58.96	50.04	33.20	63.93	68.35	53.57
	RoPT	70	40.78	53.03	60.38	53.89	34.40	66.05	71.19	54.25
	LoopPrune (ours)	70	**47.10**	**64.35**	**62.59**	**60.12**	**36.60**	**69.53**	**75.75**	**59.43**

This assignment ensures that highly-fit structures are protected from destructive mutations, while suboptimal performers undergo aggressive exploration to escape local minima.

B.4 Local Search Execution Logic

The periodic local search acts as a fine-grained hill-climbing refinement to optimize loop multiplicities. When triggered, it operates on the best-performing individual $\mathbf{c}^*$ for a maximum of 5 iterations. During each iteration, the procedure traverses the gene positions strictly sequentially from $i = 0$ to $2L - 1$.

At each position i, it evaluates the fitness impact of incrementing $(c_i + 1)$ or decrementing $(c_i - 1)$ the execution count. If a modification improves the fitness without violating the predefined maximum parameter retention constraint (η_{target}), the update is accepted greedily and immediately. The local search terminates early if a full sequential pass over the chromosome yields no further improvements.

B.5 RMSNorm Fine-tuning Configuration

After the evolutionary search phase identifies the best-performing architecture, we apply a lightweight fine-tuning procedure to adapt the RMSNorm layers to the new execution path. This step is crucial because transformer RMSNorm layers are originally trained end-to-end with their adjacent attention and FFN modules. When modules are recombined or iteratively reused through our loop-based execution, these normalization layers encounter out-of-distribution activations, leading to suboptimal performance.

Data Preparation. We tokenize the WikiText-2 training sets into continuous token sequences, then slice them into non-overlapping windows of length 2048 tokens. We use 1024 training windows for the fine-tuning process.

Training Configuration. The fine-tuning runs with the following hyperparameters:

- Learning rate: 1×10^{-4} with AdamW optimizer
- Weight decay: 0.01

Table 9: Zero-shot accuracy evaluation under extreme compression constraints (50% parameter retention) across all model scales and families.

Model	Method	Zero-shot Accuracy ↑							
		ARC-c	ARC-e	Wino	Hella	OBQA	PIQA	BoolQ	Avg
LLaMA2 -7B	Dense	46.25	74.62	69.06	76.01	44.20	79.11	77.68	66.70
	RPGE	24.91	27.69	50.12	27.68	25.80	51.47	50.49	36.88
	RoPT	25.68	29.84	51.14	29.84	26.60	53.16	51.38	38.23
	LoopPrune (ours)	**27.47**	**34.39**	**52.01**	**34.42**	**28.00**	**55.39**	**52.39**	**40.58**
LLaMA2 -13B	Dense	49.06	77.44	71.98	79.39	45.20	80.52	80.55	69.16
	RPGE	26.37	28.83	51.14	32.50	26.40	52.50	51.53	38.47
	RoPT	27.22	31.44	52.09	33.86	27.40	54.24	52.45	39.81
	LoopPrune (ours)	**28.07**	**35.10**	**53.04**	**36.23**	**28.60**	**56.53**	**53.46**	**41.58**
Qwen2.5 -7B	Dense	51.11	77.36	72.85	78.93	47.20	79.71	84.68	70.26
	RPGE	25.85	27.10	50.12	26.97	26.40	50.44	50.28	36.74
	RoPT	27.65	33.50	52.49	32.61	28.60	53.86	53.52	40.32
	LoopPrune (ours)	**30.29**	**40.87**	**55.41**	**40.29**	**32.20**	**58.11**	**57.19**	**44.91**
Qwen2.5 -14B	Dense	58.87	79.17	75.37	82.90	45.00	82.10	85.23	72.66
	RPGE	26.37	27.65	51.14	27.52	23.60	51.47	51.31	37.01
	RoPT	29.52	34.68	53.67	32.84	26.60	54.19	53.82	40.76
	LoopPrune (ours)	**33.96**	**42.00**	**57.22**	**41.11**	**31.00**	**59.30**	**58.35**	**46.13**
LLaMA3.1 -8B	Dense	53.41	81.14	73.32	78.92	44.80	81.23	82.08	70.70
	RPGE	30.98	31.48	37.79	55.33	37.30	29.60	60.23	40.39
	RoPT	26.01	34.30	43.43	57.70	42.70	31.60	57.13	41.84
	LoopPrune (ours)	**37.20**	**49.03**	**59.98**	**48.10**	**33.40**	**63.38**	**63.73**	**50.69**
Qwen3 -8B	Dense	56.40	80.93	67.88	74.93	41.40	77.75	86.61	69.41
	RPGE	34.47	36.11	53.59	34.92	28.20	58.38	57.28	43.28
	RoPT	31.23	41.92	55.41	40.11	30.00	55.50	61.07	45.03
	LoopPrune (ours)	**37.80**	**47.73**	**57.30**	**45.30**	**31.60**	**61.26**	**64.89**	**49.41**
Qwen3.5 -9B	Dense	55.80	74.16	73.09	78.07	42.60	80.30	89.17	70.46
	RPGE	26.62	35.56	51.54	36.43	27.20	54.46	58.72	41.50
	RoPT	34.13	48.15	55.41	50.15	30.20	60.12	66.18	49.19
	LoopPrune (ours)	**40.10**	**56.65**	**58.41**	**59.35**	**32.60**	**66.59**	**71.80**	**55.07**

- Learning rate schedule: Linear warmup for 100 steps, followed by linear decay
- Gradient clipping: Maximum norm of 1.0
- Processing: One window per iteration (effective batch size of 1)

Mixed Precision Training. To ensure numerical stability while maintaining memory efficiency, we employ a mixed precision strategy. RMSNorm layers are kept in FP32 precision, while all other model parameters remain in FP16. Automatic mixed precision (AMP) is enabled during both forward and backward passes, allowing FP32 RMSNorm computations while other operations use FP16.

Model Selection. We evaluate the model on the validation set after each epoch and save the checkpoint with the lowest perplexity. This best checkpoint is then used for final evaluation on downstream tasks.

C Inference Efficiency at Scale and KV-Cache Behavior

Characterizing real-time deployment requires evaluating beyond batch size 1, particularly regarding potential KV-cache multiplication during repeated attention executions. As evaluated in Table 5, our baseline latency metrics utilize a relatively short sequence length ($L_{\text{in}} = 128, L_{\text{out}} = 128$). In this regime, the overall memory footprint is heavily dominated by the Fixed Overhead (Model Parameters + Framework Context), while the per-sample Dynamic Overhead (KV Cache and Activations) scaling with the batch size is remarkably small.

Table 10: Advanced generative reasoning evaluation on Qwen3.5-9B. LoopPrune natively preserves algorithmic rigor significantly better than static pruning baselines.

Method	Target Retention	HumanEval (%)	MBPP (%)	MATH500 (%)
Dense (Original)	100%	70.73	56.00	55.60
LLM-Pruner	70%	28.66	22.40	21.00
SliceGPT	70%	40.85	31.60	30.20
ShortGPT	70%	45.12	33.40	35.00
RPGE	70%	38.41	29.80	28.20
RoPT	70%	36.59	26.80	28.40
LoopPrune (Ours)	70%	**51.83**	**40.20**	**39.20**
LLM-Pruner	50%	16.46	17.20	17.60
SliceGPT	50%	18.90	17.80	17.20
ShortGPT	50%	24.39	20.60	19.20
RPGE	50%	19.51	16.40	14.20
RoPT	50%	21.95	15.80	15.80
LoopPrune (Ours)	50%	**34.76**	**26.80**	**25.20**

Algorithm 1 Chromosome Decoding Strategy

Require: Chromosome $\mathbf{c} \in \{0, \ldots, K\}^{2L}$
Ensure: Execution path π
$\pi \leftarrow []$, $i \leftarrow 0$
while $i < 2L$ **do**
  if $c_i = 0$ **then**
    $i \leftarrow i + 1$ ▷ Skip module
  else
    if $c_i = 1$ **then**
      Append $\mathcal{M}_i$ to π ▷ Single execution
      $i \leftarrow i + 1$
    else
      $v \leftarrow c_i$ ▷ $c_i \geq 2$: Loop Block Start
      Identify span $[i, j]$ such that $\forall x \in [i, j], c_x = v$
      $\mathcal{B} \leftarrow [\mathcal{M}_i, \ldots, \mathcal{M}_j]$
      for $step = 1$ **to** v **do**
        Append $\mathcal{B}$ to π ▷ Repeat block v times
      end for
      $i \leftarrow j + 1$
    end if
  end if
end while
return π

C.1 Memory Footprint Scaling & KV-Cache Sharing

Because LoopPrune drastically cuts the massive fixed parameter footprint (reducing FP16 parameters from ∼138 GB in the dense LLaMA2-70B model to ∼68.7 GB at 50% retention), our model provides immense scaling headroom. Table 11 illustrates the projected memory distribution for LLaMA2-70B across larger batch sizes.

Furthermore, to prevent the KV-cache from duplicating linearly across iterations when an attention block is looped, we avoid naive memory allocation. We implement dynamic pointer sharing for the static components of the KV states across the looped module iterations. This effectively bounds the memory footprint, ensuring stability and highly competitive throughput even at extended context lengths and larger batch sizes.

Algorithm 2 The Complete End-to-End LoopPrune Pipeline

Require: Dense LLM $\mathcal{M}$, Calibration Data $\mathcal{D}_{cal}$, Generations G.
Ensure: Compressed LLM $\mathcal{M}^*$.

// 1. Initialization Phase
$P_0 \leftarrow \text{MultiTierInit}(\mathcal{M}, \mathcal{D}_{cal})$
EvaluateFitnessAndConstraints(P_0)
$c_{best} \leftarrow \text{GetBest}(P_0)$
// 2. Evolutionary Search
for $g = 1$ **to** G **do**
  $P_{new} \leftarrow \emptyset$
  while $|P_{new}| < |P_{g-1}|$ **do**
    $p_1, p_2 \leftarrow \text{Selection}(P_{g-1})$
    $c_1, c_2 \leftarrow \text{Crossover}(p_1, p_2)$
    $c_1, c_2 \leftarrow \text{MultiStrategyMutation}(c_1, c_2)$
    $P_{new} \leftarrow P_{new} \cup \{c_1, c_2\}$
  end while
  EvaluateFitnessAndConstraints(P_{new})
  if IsPeriodicTrigger(g) **then**
    $c_{local} \leftarrow \text{LocalSearch}(\text{GetBest}(P_{new}))$ ▷ *Periodic Local Search*
    $P_{new} \leftarrow \text{UpdateElite}(P_{new}, c_{local})$
  end if
  $P_g \leftarrow P_{new}$
  $c_{best} \leftarrow \text{UpdateBest}(c_{best}, P_g)$
end for
// 3. Execution & Fine-Tuning
$\mathcal{M}_{pruned} \leftarrow \text{PathDecoding}(\mathcal{M}, c_{best})$ ▷ *Extract Best Individual*
$\mathcal{M}^* \leftarrow \text{RMSNormFineTuning}(\mathcal{M}_{pruned}, \mathcal{D}_{cal})$
return $\mathcal{M}^*$

Table 11: Projected memory distribution for LLaMA2-70B across larger batch sizes ($L_{\text{in}} = 128, L_{\text{out}} = 128$, FP16). Fixed memory includes weights and framework context.

Method (LLaMA2-70B)	Batch Size	Fixed Memory	Dynamic KV/Act (est.)	Total VRAM Footprint
Dense (Original)	BS = 1	∼151.0 GB	∼0.9 GB	151.9 GB
Dense (Original)	BS = 8	∼151.0 GB	∼7.2 GB	158.2 GB
Dense (Original)	BS = 16	∼151.0 GB	∼14.4 GB	165.4 GB
LoopPrune (50%)	BS = 1	∼77.0 GB	∼0.5 GB	**77.5 GB**
LoopPrune (50%)	BS = 8	∼77.0 GB	∼4.0 GB	**81.0 GB**
LoopPrune (50%)	BS = 16	∼77.0 GB	∼8.0 GB	**85.0 GB**

C.2 Decoding Throughput Scaling

To understand the complete system-level behavior of our dynamic execution path, we evaluate the exact inference throughput (measured in System Tokens / Second during the autoregressive decoding phase) across larger batch sizes on LLaMA2-70B.

Table 12: System decoding throughput comparison (Tokens / Second) on LLaMA2-70B across larger batch sizes.

Method (LLaMA2-70B)	Target Retention	BS = 1 (Tok/s)	BS = 8 (Tok/s)	BS = 16 (Tok/s)
Dense (Original)	100%	17.92	132.6	248.5
SliceGPT (Baseline)	50%	21.79	148.3	263.2
LoopPrune (Ours)	50%	**27.58**	**212.4**	**405.6**

As the results in Table 12 demonstrate, LoopPrune not only maintains a highly competitive throughput at Batch Size 1, but its advantage actually widens significantly as the batch size scales up:

- **Memory-Bandwidth Bound Decoding:** In modern LLM inference, the decoding phase is heavily memory-bandwidth bound. Because LoopPrune strictly removes unique parameters from memory (drastically reducing the massive weight-loading overhead) without introducing any auxiliary transformation layers, its throughput scales exceptionally well.
- **Overcoming Auxiliary Rotation Bottlenecks:** Standard structured pruning baselines like SliceGPT require executing heavy orthogonal rotation matrices that cannot be pre-computed. As the batch size increases, the computational overhead of these dense rotation operations acts as a severe bottleneck. Consequently, SliceGPT's relative speedup over the Dense model noticeably degrades at BS=16, whereas LoopPrune sustains a robust $\sim$1.6$\times$ speedup.

LoopPrune's ability to cleanly decouple effective depth from the physical memory footprint ensures that we do not trade off latency for reasoning capacity, offering a superior and highly practical Pareto frontier between real-world system throughput and intelligence.

D Further Ablation Studies

D.1 Impact of the Iterative Loop Mechanism

To explicitly validate the core contribution of our proposed iterative execution paradigm, we conduct an ablation study targeting the loop mechanism itself. By restricting the maximum gene multiplicity K to 1 during the evolutionary search, we disable any iterative block execution. This forces the framework to degenerate into a standard "Skip-Only" module selection process that strictly matches standard static pruning (Iso-FLOP and Iso-Latency).

D.1.1 Deconstructing Execution: Parameter-Compute Trade-off

Optimizing for unique parameters decouples the memory footprint from the executed compute steps. To transparently characterize this efficiency trade-off, we isolate the gains of our search methodology from the overhead of extra executed compute.

Table 13: Deconstructing execution efficiency on WikiText-2. "Ours (Skip-Only)" matches FLOPs/latency with RoPT, while "LoopPrune (Full)" matches unique parameters.

Method	Target Retention	Effective Depth	Executed FLOPs / Latency	PPL
Dense (Original)	100%	40	1.00$\times$	4.90
RoPT (Baseline)	70%	28	$\sim$0.70$\times$	11.75
Ours (Skip-Only)	70%	28	$\sim$0.70$\times$ (Iso-FLOP & Iso-Latency)	7.43
LoopPrune (Full)	70%	36	$\sim$0.90$\times$ (Iso-Parameter vs. RoPT)	**6.59**
RoPT (Baseline)	50%	20	$\sim$0.50$\times$	21.05
Ours (Skip-Only)	50%	20	$\sim$0.50$\times$ (Iso-FLOP & Iso-Latency)	11.98
LoopPrune (Full)	50%	31	$\sim$0.78$\times$ (Iso-Parameter vs. RoPT)	**10.03**

As detailed in Table 13, when restricted to strictly sequential execution, our *Skip-Only* variant massively outperforms the baseline (e.g., reducing PPL from 11.75 to 7.43 at 70% retention). This definitively proves the superiority of our dynamic execution-path formulation, entirely independent of any loop-induced compute. Furthermore, the full loop mechanism is essential for fully decoupling depth from parameter count, further dropping the Iso-Parameter PPL to 10.03 at 50% retention while naturally bounding the dynamic compute overhead.

D.1.2 Impact on Zero-Shot Reasoning

As detailed in Table 14, restricting the model to a strictly sequential execution path (*Ours (Skip-Only)*) leads to noticeable degradations in overall capacity compared to the *LoopPrune (Full)* model. Without the ability to dynamically reuse pre-trained primitives, the average zero-shot reasoning accuracy drops by 1.38 points. By restoring the lost depth through optimized loops, the full model achieves superior performance on five out of seven reasoning tasks.

Importantly, even when completely stripped of the iterative loop mechanism, our *Skip-Only* variant still significantly outperforms the strongest baseline, RoPT, by a wide margin (e.g., yielding a 4.32

Table 14: Ablation on downstream reasoning tasks for the iterative loop mechanism using LLaMA2-13B at 70% parameter retention. The "LoopPrune (Full)" configuration effectively reclaims reasoning depth without introducing new parameters.

Method	Wiki2↓	Zero-shot Accuracy↑							
		ARC-c	ARC-e	Wino	Hella	OBQA	PIQA	BoolQ	Avg
RoPT	11.75	32.17	52.65	54.14	59.45	35.20	70.46	57.16	51.60
Ours (Skip-Only)	7.43	31.40	59.68	52.80	67.11	34.40	74.54	59.30	54.18
LoopPrune (Full)	**6.59**	**34.30**	**61.36**	**55.49**	66.09	**37.60**	**75.08**	58.99	**55.56**

PPL reduction and a 2.58% gain in average accuracy). This secondary finding highlights that our specialized evolutionary search and multi-tier initialization strategies are highly effective in their own right, discovering vastly superior sub-networks before the loop mechanism even amplifies them.

D.2 Allocation and Synergy of Initialization Strategies

D.2.1 Empirical Justification

To ground our population allocation in empirical data, we first analyze the raw quality of seed individuals generated independently by the three non-elite strategies. By randomly sampling 640 individuals for each method on LLaMA2-7B, we evaluate their zero-shot perplexity prior to evolutionary refinement.

As depicted in Figure 4, the *Wanda Guided* strategy demonstrates the most significant advantage, achieving the lowest perplexity and the tightest variance by protecting critical modules. The *Structural Prior* follows, providing a stable macroscopic heuristic. Conversely, the *Constrained Random* approach yields the poorest overall quality with an extremely high variance, underscoring the fragility of the loop-enabled search space when relying on unguided sampling.

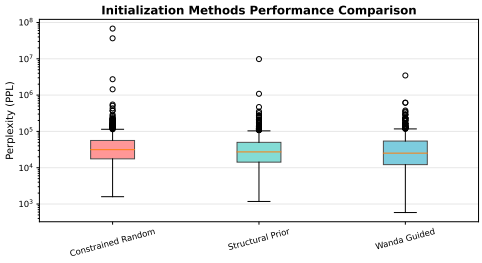


Figure 4: Box plots of zero-shot perplexity (PPL) distributions for different initialization strategies on LLaMA2-7B. Each method is evaluated across 640 randomly generated seed individuals. The y-axis is on a logarithmic scale.

Guided by these performance distributions, alongside the proven superiority of elite individuals demonstrated earlier, we explicitly partition the starting population to balance strong exploitation with necessary genetic diversity. Consequently, the initialization employs a hierarchical split: 60% is allocated to elite transfer to ensure rapid convergence, 20% to Wanda-guided and 15% to structural priors reflecting their robust empirical quality, and a residual 5% to constrained random to inject diversity while preventing overall population degradation. Furthermore, to complement this initialization, Local Search is triggered every 10 generations for fine-grained refinement.

D.2.2 Synergy of Initialization Components

Table 15: Ablation study of initialization strategies on LLaMA2-13B at 50% sparsity. Each row represents the incremental addition of a specific initialization prior. The Gain is calculated relative to the Pure Random baseline.

Configuration	**Initialization Strategy Breakdown**	**Final PPL**	**Gain**
Pure Random	100% Random	21.25	-
+ Wanda Guidance	80% Random + 20% Wanda	20.84	+1.9%
+ Structural Priors	65% Random + 20% Wanda + 15% Structural	20.51	+3.5%
Full Mix (Ours)	5% Random + 20% Wanda + 15% Structural + 60% Elite Transfer	**17.27**	**+18.7%**

Beyond the isolated quality of initial seeds, we further investigate how these components synergize to enhance the final evolutionary outcome. Using LLaMA2-13B at a 50% parameter retention ratio, we

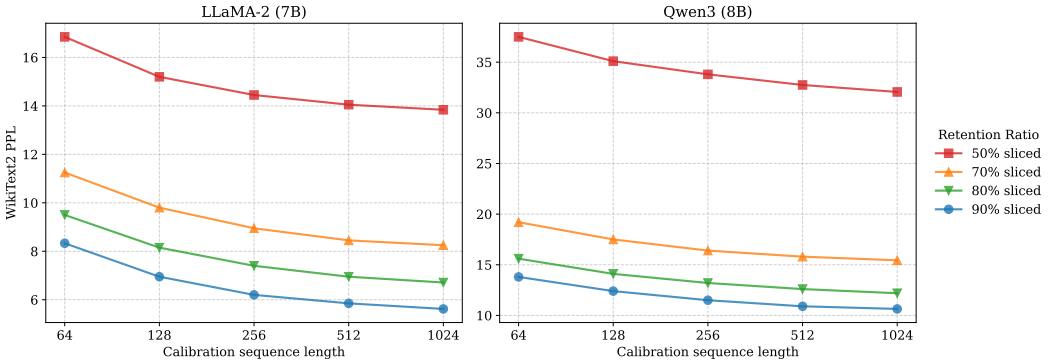


Figure 5: The effect of the calibration sequence length on the WikiText-2 perplexity. Results are shown for LLaMA2-7B (left) and Qwen3-8B (right) across multiple parameter retention ratios (50%, 70%, 80%, and 90%). Longer calibration sequences consistently improve final model quality.

start from a purely random baseline and incrementally integrate our proposed heuristic priors and elite transfer mechanisms.

As reported in Table 15, relying exclusively on random sampling (*Pure Random*) yields a suboptimal final perplexity of 21.25, indicating that the evolutionary search struggles to efficiently navigate the vast loop-enabled space without structural guidance. Introducing 20% Wanda-guided individuals provides an initial 1.9% gain by protecting highly sensitive modules. The further inclusion of 15% Structural Priors establishes a more robust macroscopic topology, yielding a cumulative 3.5% improvement over the baseline.

Crucially, the most substantial performance leap occurs with the integration of 60% Elite Transfer (*Full Mix*). This addition drops the perplexity significantly to 17.27, representing an 18.7% total gain. These results clearly demonstrate that while heuristic and data-driven priors (Structural and Wanda) provide necessary foundational stability, their individual improvements are relatively modest compared to Elite Transfer. The cross-model structural knowledge inherited through historical elites acts as the primary driver for discovering top-tier execution paths.

D.3 Sensitivity to Calibration Data and Hyperparameters

While LoopPrune fundamentally relies on evolutionary architecture discovery, the fitness evaluation during the search phase necessitates a calibration dataset. To thoroughly investigate the robustness and sensitivity of LoopPrune to these calibration settings, we conducted an ablation study on the influence of calibration sequence length and the choice of calibration dataset.

Effect of Sequence Length. The sequence length of the calibration data directly dictates the context window available for the model to minimize its calibration loss. Figure 5 illustrates the WikiText-2 perplexity of the compressed LLaMA2-7B and Qwen3-8B models across varying calibration sequence lengths (from 64 to 1024) at multiple sparsity levels (50%, 70%, 80%, and 90% retention). As expected, increasing the calibration sequence length consistently yields better (lower) perplexity across all configurations. The performance gain is particularly pronounced when scaling from extremely short sequences (e.g., 64) to moderate lengths (e.g., 256). Beyond a sequence length of 512, the marginal improvement begins to saturate. Consequently, to balance computational overhead during the evolutionary search with optimal final performance, we adopted a sequence length of 1024 for our main experiments. It is also notable that higher sparsity levels (e.g., 50% retention) exhibit a higher absolute sensitivity to truncated context lengths.

Influence of Calibration Dataset. We further examined whether the domain of the calibration data biases the final compressed model. Table 16 compares the WikiText-2 test perplexity when LoopPrune is calibrated using the PTB (Marcus et al., 1993) dataset versus the WikiText-2 training set (both using a fixed sequence length of 1024). Calibrating on PTB leads to a slight but consistent increase in WikiText-2 test perplexity compared to calibrating on the in-domain WikiText-2 dataset. For instance, at 70% retention, calibrating on PTB yields a perplexity of 8.42, while WikiText-2 calibration achieves 8.25. This indicates that while in-domain calibration provides a marginal

Table 16: The effect of different calibration datasets (PTB vs. WikiText-2) on the final WikiText-2 test perplexity of LLaMA2-7B. The calibration sequence length is fixed at 1024.

Calibration Dataset	Retention Ratio			
	90%	**80%**	**70%**	**50%**
PTB	5.72	6.84	8.42	14.15
WikiText-2	5.62	6.71	8.25	13.84

advantage, LoopPrune remains highly robust to cross-domain calibration, successfully identifying universally effective iterative execution paths rather than heavily overfitting to the specific calibration distribution.

D.4 Impact of Genetic Selection and Crossover Strategies

While selection and crossover are indispensable operations in any Genetic Algorithm, their specific formulations dramatically alter the search efficiency within our loop-enabled topological space. To evaluate their impact, we incrementally "unpack" our optimal configuration (Top-20 selection with Two-point crossover, denoted as the *Baseline GA* in Table 6) starting from a naive vanilla setup.

Selection Mechanism: Top-20. In a standard GA, selection is often fully random or uses weak tournament pressure. Our *Top-20 selection* strategy strictly truncates the mating pool to only the top 20% of valid individuals ranked by fitness. Two distinct parents are uniformly sampled exclusively from this elite fraction. Given the vastness of the loop-multiplicity search space (K^{2L}), this aggressive truncation heavily biases the search toward exploiting high-performing structural motifs, preventing regression.

Crossover Mechanism: Single-point vs. Two-point. Crossover dictates how parent chromosomes exchange genetic material to form offspring.

- *Single-point crossover* randomly selects one split index; the first child inherits the prefix from Parent A and the suffix from Parent B (and vice versa for the second child).
- *Two-point crossover* selects two distinct indices, defining a contiguous middle segment. The parents exchange only this middle segment.

Because Large Language Models exhibit strong block-wise phase transitions (e.g., embedding-heavy early layers, reasoning-heavy middle layers, and projection-heavy late layers), Two-point crossover is structurally superior. It allows the algorithm to swap a continuous functional "middle" block while perfectly preserving the synergistic "front" and "back" modules of the parents, whereas Single-point crossover aggressively severs these end-to-end dependencies.

Table 17: Ablation of genetic operators on LLaMA2-13B at 50% parameter retention. We incrementally upgrade a Vanilla GA towards our optimal baseline configuration.

Method	**Operator Configuration**	**PPL**
Vanilla GA	Random Selection + Single-point Crossover	29.21
+ Top-20 Selection	Top-20 Selection + Single-point Crossover	20.50
+ Two-point Crossover (Ours)	Top-20 Selection + Two-point Crossover	19.24

As shown in Table 17, the Vanilla GA yields a poor perplexity of 29.21. The introduction of Top-20 selection drives a massive improvement, dropping the PPL to 20.50 by actively propagating top-tier architectures. Replacing Single-point with Two-point crossover further refines the search, pushing the PPL down to 19.24 by preserving continuous architectural blocks during recombination.

D.5 Ablation of GA Hyperparameters

To optimize the search efficiency of LoopPrune, we perform an extensive hyperparameter ablation study on LLaMA2-13B at a 50% parameter retention ratio. We analyze the impact of selection

Table 18: Impact of GA hyperparameters on the final search performance (WikiText-2 PPL) for LLaMA2-13B at 50% sparsity. Bold indicates our selected optimal configuration.

Hyperparameter	Search Space	Config.	Final PPL
Selection Pressure (Top-N)	Exploitation vs. Diversity	Top-10	17.89
		Top-20	**17.27**
		Top-30	18.94
Crossover Rate (p_c)	Global Exploration	90%	19.04
		100%	**17.27**
Mutation Rate (p_m)	Local Refining / Perturbation	0.1	18.70
		0.05	**17.27**

pressure, crossover rate, and mutation rate on the final convergence quality, as measured by WikiText-2 perplexity.

As reported in Table 18, the search efficiency of LoopPrune is highly sensitive to the balance between exploration and exploitation governed by these operators.

Selection Pressure. We evaluate truncation selection using the top-10, top-20, and top-30 individuals. The *Top-20* configuration achieves the optimal perplexity (17.27) by striking the right balance. An overly aggressive selection (Top-10) causes the population to lose diversity, prematurely converging to a suboptimal local minimum (17.89). Conversely, a weaker selection pressure (Top-30) fails to effectively propagate high-quality structural motifs, leading to a noticeable performance drop (18.94).

Crossover Rate. Maintaining a 100% crossover rate ($p_c = 1.0$) proves superior to a reduced rate of 90% (19.04). In the vast and non-linear loop-enabled search space (K^{2L}), continuous recombination is essential for discovering and assembling efficient macroscopic execution patterns across different semantic blocks.

Mutation Rate. We observe that a lower mutation rate of 0.05 yields the best results. Increasing the mutation rate to 0.1 causes the perplexity to degrade to 18.70. This degradation indicates that at high sparsity levels, excessive random mutations tend to disrupt delicately balanced execution paths. Therefore, maintaining a low mutation rate ensures that local perturbations safely refine, rather than destroy, the established functional sub-structures.

D.6 Post-Training Alignment Stability

Standard static pruning often catastrophically disrupts the delicate instruction-following templates of aligned models. To evaluate alignment stability under high sparsity, we conducted preliminary MT-Bench evaluations on Qwen2.5-7B-Instruct utilizing multi-turn dialogue data for our conversational RMSNorm recalibration.

As demonstrated in Table 19, LoopPrune exhibits a clear structural advantage. At an aggressive 60% retention, static pruning (RoPT) suffers from catastrophic forgetting (dropping to 4.15), whereas LoopPrune degrades gracefully, maintaining a competitive score of 6.02. This confirms our theoretical intuition: alignment relies on semantic directional geometry. Iterative looping primarily alters the magnitude/variance of the residual stream—which our conversational RMSNorm recalibration explicitly corrects—rather than destroying the underlying semantic direction. This structurally preserves instruction-following capabilities without full-parameter RLHF retraining.

Table 19: Post-training alignment stability evaluated on MT-Bench using Qwen2.5-7B-Instruct.

Method	Target Retention	MT-Bench Score
Dense (Original)	100%	8.25
RoPT (Baseline)	80%	6.78
LoopPrune (Ours)	80%	**7.54**
RoPT (Baseline)	60%	4.15
LoopPrune (Ours)	60%	**6.02**

E Architecture Analysis

Table 20: Top-3 elite chromosomes for different model configurations.

Model	Sparsity	Layers	Rank	Chromosome	PPL w.o. FT
Qwen3.5 -9B	30%	32	#1	[1,1,1,0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2, 1,2,1,2,1,1,1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,0,1,1,1,1,1]	19.79
			#2	[1,1,1,0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2, 1,2,1,2,1,1,1,0,1,0,1,0,1,0,1,0,1,1,0,2,1,0,1,1,1,1,1]	19.94
			#3	[1,1,1,0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2, 1,2,1,2,1,1,1,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,0,1,1,1,1,1]	20.01
LLaMA2 -13B	50%	40	#1	[1,2,2,2,1,0,1,2,2,2,0,0,1,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2, 2,2,2,2,2,2,2,2,2,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0 ,1,0,1,0,1,1,1,1,1]	17.27
			#2	[1,2,2,2,1,0,1,2,2,2,0,0,1,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2, 2,2,2,2,2,2,2,2,2,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0 ,0,0,1,0,1,1,1,1,1]	17.36
			#3	[1,2,2,2,1,0,1,2,2,2,0,0,1,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2, 2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0 ,0,1,1,0,1,1,1,1,1]	17.42
LLaMA2 -70B	50%	80	#1	[1,1,2,2,2,2,2,2,1,1,0,0,0,1,2,1,2,2,2,2,2,0,0,0,0,1,1,2,2,1,1,2,2,2,2,2, 2,2,2,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2 ,2,2,2,2,2,2,2,2,0,2,2,2,2,2,2,2,2,2,2,0,1,2,0,0,0,0,0,0,1,0,0,1,0,0, 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,1,0 ,0,0,1,1,1,1,1,0,0,1,1,1,1,1,1,2,1,1,1]	10.90
			#2	[1,1,2,2,2,2,2,2,1,0,0,0,1,1,1,2,2,2,2,2,0,0,0,0,1,1,2,2,1,0,2,2,2,2,2, 2,2,2,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2 ,2,2,2,1,2,1,2,2,0,2,2,2,2,2,2,2,2,2,2,0,1,2,0,0,0,0,1,0,1,1,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,2,1 ,0,0,1,1,1,1,0,0,1,1,1,1,1,1,2,1,2,1]	11.54
			#3	[1,1,2,1,2,2,2,2,1,1,0,0,1,2,1,2,2,2,2,2,0,0,0,0,1,1,2,0,0,1,2,2,2,2,2, 2,2,2,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,2,2,2,1,2,2,2 ,2,2,2,2,2,2,2,2,0,2,2,2,2,2,2,2,2,2,2,0,1,2,0,0,0,0,0,0,0,1,0,0,0,1,0,0, 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,1,0 ,0,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1]	11.55

We analyze the discovered architectures to understand what execution patterns LoopPrune favors. This section presents representative elite individuals and statistical analysis of gene value distributions across different models and sparsity levels.

E.1 Individual Examples

Table 20 shows the Top-3 elite chromosomes discovered for three representative configurations. Each chromosome encodes the execution pattern for all modules, where gene values indicate: 0=skip, 1=execute once, 2=execute twice, 3=execute three times, and so on. We report the perplexity before RMSNorm fine-tuning to show the raw performance of discovered architectures.

Observed patterns: (1) *Front-back preservation*: Early layers and late layers consistently retain high gene values, while middle layers show aggressive pruning. (2) *Loop placement*: Loop execution concentrates in front and back regions, with occasional loops in middle layers. (3) *Sparse middle*: At 50% sparsity, middle layers exhibit long sequences of skipped modules, indicating these layers contribute less to final performance. (4) *Rare high loops*: Values strictly greater than two appear infrequently, suggesting double execution suffices for most modules.

E.2 Gene Value Distribution Statistics

To quantify the prevalence of different execution modes, we analyze the gene value distribution across Top-60 elite individuals from three representative configurations. Table 21 reports the percentage of each gene value across all modules in the elite pool.

Table 21: Gene value distribution statistics across Top-60 elite individuals. Values represent the percentage of genes with each execution mode: 0 (skip), 1 (single), 2 (double loop), 3+ (more loops). Higher sparsity levels show increased skip rates and reduced loop usage.

Model	Sparsity	0 (Skip)	1 (Single)	2 (Loop)	3+	Total Genes
Qwen3.5-9B	30%	42.2%	33.6%	24.2%	0.0%	3,840
LLaMA2-13B	50%	52.3%	21.6%	25.9%	0.1%	4,800
LLaMA2-70B	50%	49.5%	23.2%	27.3%	0.0%	9,600

Key observations: (1) *Sparsity-dependent skip rate*: At 30% sparsity, 42.2% of genes are skipped, whereas at 50% sparsity, the skip rate increases to roughly 50%. (2) *Balanced single vs. loop*: Single execution and loop execution are roughly balanced, with loop execution accounting for a significant portion of all genes. (3) *Rare high-order loops*: Multiplicities of three or higher constitute a negligible fraction of genes ($\leq 0.1\%$), validating our encoding design that simple execution patterns dominate. (4) *Cross-model consistency*: LLaMA2-13B and LLaMA2-70B at the same 50% sparsity exhibit nearly identical distributions, suggesting discovered patterns generalize across model scales.

These statistics confirm that LoopPrune's evolutionary search converges to interpretable, structured execution patterns rather than random configurations, with clear preferences for front-back preservation and selective loop placement.

F Theoretical Justification for Loop Patterns

To formally understand why LoopPrune effectively preserves reasoning capacity and naturally discovers specific architectural patterns (e.g., dense boundary loops and sparse middles, as empirically observed in Appendix E), we provide a justification rooted in the Ordinary Differential Equation (ODE) view of residual networks.

ODE Formulation of Iterative Execution. A transformer's forward pass can be modeled as an Euler discretization of an ODE. Let x_ℓ be the hidden representation at layer ℓ, and f_ℓ be the residual transformation (Attention + FFN). The standard sequential update is $x_{\ell+1} = x_\ell + f_\ell(x_\ell)$. When LoopPrune applies an iterative loop of multiplicity k to a specific retained module i, the local discrete dynamical system becomes $x_{i,t} = x_{i,t-1} + f_i(x_{i,t-1})$ for $t = 1, \ldots, k$.

Error Bound in Middle Layers (ϵ-Smoothness Region). Recent literature shows that middle layers of LLMs exhibit high representation similarity, meaning they reside in a flat functional basin.

Definition F.1 (ϵ-Smoothness)**.** A sequence of sequential layers from i to $i + k - 1$ is ϵ-smooth if the functional difference between adjacent layers is bounded: $\|f_{\ell+1}(x) - f_\ell(x)\| \leq \epsilon$ for all valid x.

Theorem F.2 (Error Bound of Looped Execution)**.** *Assume f_i is L-Lipschitz continuous. If layers i to $i + k - 1$ are ϵ-smooth, replacing the sequential execution with a k-step loop of module i yields a representation $\tilde{x}_{i+k}$ that approximates the original dense execution x_{i+k} with an error bounded by:*

$$\|x_{i+k} - \tilde{x}_{i+k}\| \leq \epsilon \sum_{j=1}^{k} (1 + L)^{j-1} \tag{6}$$

Proof Sketch. For a single substitution step ($k = 2$), the dense network computes $x_{i+2} = x_{i+1} + f_{i+1}(x_{i+1})$. The looped network computes $\tilde{x}_{i+2} = x_{i+1} + f_i(x_{i+1})$. The deviation is exactly $\|f_{i+1}(x_{i+1}) - f_i(x_{i+1})\|$, which is bounded by ϵ due to ϵ-smoothness. By induction over k steps with an L-Lipschitz condition, the accumulated error across the sequence is bounded by the geometric series $\epsilon \sum_{j=1}^{k} (1 + L)^{j-1}$.

Capacity Expansion in Boundary Layers (Phase Transitions). Unlike middle layers, front and back layers perform heavy semantic lifting (e.g., embedding-to-manifold and context-to-logits). In these boundary regions, the ϵ-smoothness assumption breaks down (ϵ is large), meaning representations undergo high-curvature transformations. To traverse these regions without the parameter cost of the dense sequence, LoopPrune dynamically assigns loop multiplicities $k \geq 2$ to highly expressive boundary modules. This effectively expands the function class of that module from $f_i(x)$ to the compositional space $f_i \circ \cdots \circ f_i(x)$. Therefore, iterative execution acts as an expressivity amplifier,

providing the necessary non-linear depth to cross semantic phase transitions while strictly adhering to the compressed parameter budget.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the core contribution of Depth-Parameter Decoupling and the evolutionary search method, which perfectly align with the experimental evaluations provided in subsequent sections.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the challenge of execution-induced distribution shift (Section 3.5).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The theoretical justification for loop patterns, including the ϵ-Smoothness assumption and the error bound theorem along with its proof sketch, is provided in Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper details the hyperparameter configurations, algorithm pseudo-code, and evaluation metrics in Section 4.1 and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The code is currently withheld to maintain double-blind anonymity during the review process, but will be released as an open-source repository upon acceptance. Data used (WikiText-2, etc.) are public.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 and Appendix B (Implementation Details) clearly specify the datasets, mixed-precision settings, learning rates, and GA hyperparameters used for the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide appropriate evidence of robustness and significance through comprehensive ablation studies on all core components and consistent evaluations across diverse benchmarks.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 explicitly states that experiments were run on NVIDIA A100 80GB GPUs and provides the estimated GPU hours required for the search and fine-tuning phases.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research focuses on algorithmic improvements for model compression and strictly adheres to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: The paper primarily focuses on foundational compression algorithms.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release newly trained foundation models or scraped datasets; it utilizes existing open-source models (e.g., LLaMA, Qwen) and standard public benchmarks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing foundation models (LLaMA2/3.1, Qwen2.5/3/3.5) and datasets (WikiText-2, downstream evaluation benchmarks) are properly cited in the main text and bibliography.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper proposes a new algorithmic framework but does not introduce new datasets or foundational models as standalone assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This research does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: This research does not involve human subjects, hence no IRB approval is required.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: While the paper compresses LLMs, LLMs were not used as an active or non-standard component to generate the core evolutionary methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.