LEARNING OPTIMIZED TRANSFORMER STRUCTURE FROM GENERALIZED RESIDUAL CONSTRAINED TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are conventionally trained with a fixed, dense architecture, forcing the entire model to be optimized throughout training despite substantial structural redundancy. Post-training pruning streamlines the model structure for deployment efficiency, but it does not mitigate the training-time costs. In this paper, we propose Generalized Residual Constrained Training (GRCT), a rate–distortion training paradigm built on residual modeling that jointly optimizes task performance and model structure. By optimizing the residual constraints, GRCT directly learns an adaptive model structure during training, reducing training costs as the structure contracts and ultimately yielding a deployable compact model without an extra pruning and recovery stage. In extensive experiments, GRCT outperforms all pruning baselines at the same or higher pruning rates. Compared with the dense model, GRCT even achieves 93.8% of layer pruning while improving accuracy by 1.50 percentage points. Moreover, GRCT speeds up training and inference by up to $1.53\times$ and $4.06\times$, with memory savings of 41.1% and 66.0% on a representative task, substantially improving time and memory efficiency of both training and inference. GRCT shifts the focus of compression research from the final model to the entire training trajectory, providing a unified path toward structure learning for modern LLMs.

1 INTRODUCTION

Residual connections (He et al., 2016) have markedly improved the trainability of deep large language models (LLMs). With increasing network depth and parameter scale, LLMs have demonstrated remarkable capabilities across language understanding, content generation, complex reasoning, and code modeling. However, these advances come with substantial computational costs, memory consumption, and end-to-end latency, raising the question of whether such depth is fully utilized in tasks. Evidence from biological neural systems suggests that effective computation may involve only a relatively shallow serial depth (Suzuki et al., 2023). Moreover, task learning may further increase information redundancy in neural representations (Liu et al., 2026). These observations motivate a closer examination of structural redundancy and the development of more efficient architectures in LLMs.

In contrast to the structural plasticity observed in biological neural networks, LLM architectures are typically predetermined and remain fixed throughout training. In practice, efforts to reduce redundancy have thus primarily focused on post-training compression to improve inference efficiency. These methods estimate layer-wise importance using saliency signals such as activation similarity (Men et al., 2025), gradient information (Kim et al., 2024), or learned importance scores (Xia et al., 2022), and then skip or prune less important layers. Although saliency scores can indicate which layers are less important, they cannot determine how deep a model must be to perform a given task adequately, leaving the number of pruned layers dependent on a preset budget. It reveals two limitations in LLM training and compression: **during training**, optimization is confined to a predetermined architecture, preventing the model from adaptively adjusting its structural capacity to task demands; **during compression**, the architecture is directly pruned according to saliency scores, making it difficult for the model to adapt to abrupt structural changes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this paper, we quantify each layer's structural contribution through its residual distribution, and introduce generalized residual constraints within a unified rate-distortion objective that jointly optimizes task performance and model depth. This paradigm translates the compression of layer contributions during training into continuous physical pruning, enabling the network structure to adapt autonomously during task learning and obviating a post-training pruning stage. By progressively removing redundant structures, it gradually contracts the computation graph, thereby reducing computation and memory costs in later iterations.

The main contributions of this paper are summarized as follows:

- **Transformer Structural Learning.** We propose adaptive structure learning algorithm by incorporating a structural loss into the training objective, enabling the architecture to be learned autonomously and to yield a compact model directly from end-to-end training.
- **Compression as Pruning.** We propose a compression-driven pruning paradigm, using a unified residual signal to characterize both module contributions and pruning cost, allowing the model to gradually adapt to changing contributions until the modules are pruned.
- **Generalized Residual Modeling.** We introduce generalized residuals to unify the additive updates of prunable units across granularities, reveal their layer-dependent and batch-stable generalized Gaussian statistics, and derive a differentiable Exp-Golomb (EG) coding rate to quantify each layer's structural contribution.

2 PRELIMINARIES

2.1 GENERALIZED RESIDUALS AND PRUNING

In a standard pre-norm Transformer, the Attention and feed-forward network (FFN) sublayers each contribute an additive residual update to the hidden states:

$$\hat{h}_i = h_{i-1} + \mathrm{Attn}_i(h_{i-1}), \qquad h_i = \hat{h}_i + \mathrm{FFN}_i(\hat{h}_i), \tag{1}$$

where h_{i-1}, $\hat{h}_i$, and h_i denote the input, intermediate, and output hidden states, respectively. This additive property of residuals allows the contributions of consecutive layers to be considered together and expressed as a single update. We define *generalized layer* S to characterize this merging relation:

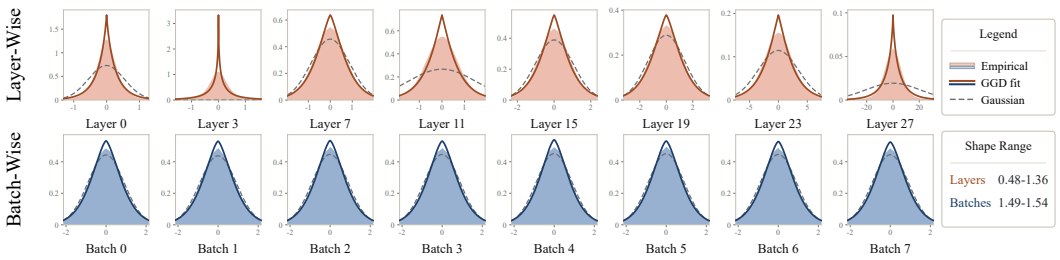
$$\widetilde{h}_u^{\mathrm{out}} = \widetilde{h}_u^{\mathrm{in}} + S_u(\widetilde{h}_u^{\mathrm{in}}), \tag{2}$$

where the additive update $S_u(\widetilde{h}_u^{\mathrm{in}})$ can be further defined as *generalized residual*. Depending on the choice of S, this definition covers an individual Attention or FFN sublayer, a full Transformer layer, or a contiguous segment of multiple layers. At the sublayer scale, it allows the task contributions of Attention and FFN to be distinguished. Keeping whole layers as units preserves the standard stacked architecture, while grouping consecutive layers can capture redundancy across layer boundaries.

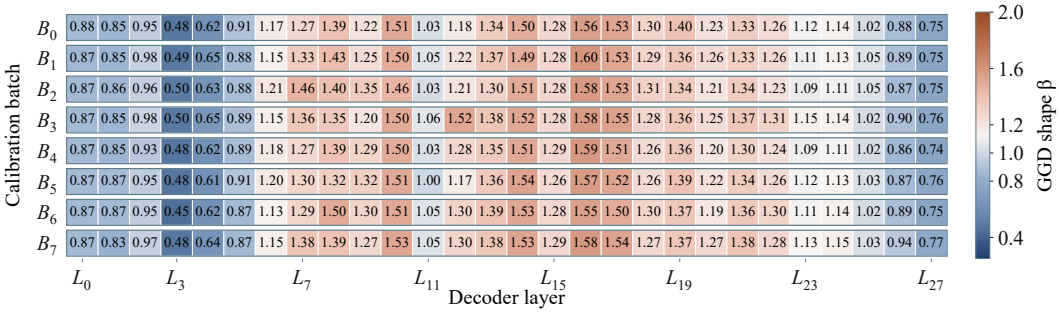
The notion of pruning can also be extended. Conventional pruning treats each layer as either retained or removed, as encoded by a binary gate $m_i \in \{0, 1\}$. Switching directly between the binary states can disrupt hidden representations and leave little opportunity for the model to adapt, while the discrete decision is also difficult to optimize alongside the model parameters. To enable a smoother transition during training, we equip the u-th generalized layer S_u with a learnable gate $g_u \in [0, 1]$ that scales its residual. A nonnegative salience function $\psi(\cdot)$ measures the significance of the residual output, and the gated residual determines whether the unit remains active:

$$\widetilde{h}_u = \begin{cases} \widetilde{h}_{u-1} + g_u\, S_u(\widetilde{h}_{u-1}), & g_u\, \psi(S_u(\widetilde{h}_{u-1})) \geq \varepsilon, \\ \widetilde{h}_{u-1}, & g_u\, \psi(S_u(\widetilde{h}_{u-1})) < \varepsilon, \end{cases} \tag{3}$$

where ε is the pruning threshold. When the gated residual remains at or above this threshold, the unit stays active; otherwise, it is replaced by the identity path and removed from the computation graph. With this generalized definition of pruning, the model can reduce a layer's contribution by suppressing its gated residual during training to achieve a learnable structure. Once the gated residual contribution becomes sufficiently small, the corresponding structure is pruned.



(a) Empirical residual densities fitted by GGDs and Gaussian distributions across layers and batches.



(b) GGD shape parameters describing residual structure with layer-wise heterogeneity and batch-wise stability.

Figure 1: GGD characteristics of residual distributions across layers and batches, exemplified by DeepSeek-R1-Distill-Qwen-7B model training on SQuAD dataset.

2.2 Generalized Residual Distribution

Reliable layer pruning requires a quantitative metric that captures either the benefit of retaining a layer or the cost of removing it. Song et al. (2024) and Kim et al. (2024) assess layer importance by measuring the loss increase caused by its removal, which requires repeated recomputation of ablation and evaluation, although it is aligned with model performance. Men et al. (2025) estimate layer importance using the cosine distance between hidden states with computational efficiency, but it only measures representational deviation and does not quantify the contribution of each layer. The generalized residual provides a more explicit representation of the additive contribution injected into the hidden-state stream by each layer transformation.

Generalized Gaussian Model. The generalized Gaussian distribution (GGD) family can be uniformly described as

$$p(r;\mu,\alpha,\beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} \exp\left[-\left(\frac{|r-\mu|}{\alpha}\right)^{\beta}\right], \tag{4}$$

where μ, α, and β denote the location, scale, and shape parameters. In particular, the shape parameter primarily determines the structural characteristics of the distribution. When $\beta = 1$ and $\beta = 2$, the GGD degrades to the Laplace and Gaussian distributions, respectively.

Generalized residuals are observed to be well modeled by the GGD family across models, tasks, and layer granularities from our extensive examinations. Fig. 1a shows that GGDs fit the residual densities more closely than a standard Gaussian distribution, with all fitted shape parameters markedly below 2. Fig. 1b further reveals the layer-wise heterogeneity and batch-wise stability of the residual distributions during model training. This observation suggests that residual statistics capture variation in individual layers' contributions while remaining robust to sampling variability across batches, supporting their use as a stable signal for structure learning.

Exp-Golomb coding. The GGD provides an intuitive characterization of residual statistics, while their structural cost needs to be further quantified to guide the structure learning. Motivated by the close match between Exp-Golomb (EG) codes and quantized GGD sources with small shape parameters (Wen & Villasenor, 1999), we define the residual rate using order-0 EG code lengths, thereby translating this characterization into an optimizable training objective.

For each residual element $r_{u,j}$, we quantize its magnitude as $q_{u,j} = \mathrm{round}(2^b|r_{u,j}|)$ and map it to the signed index $z_{u,j} = 2q_{u,j} + \mathbb{I}[r_{u,j} < 0]$. Its order-0 EG code length is

$$\ell_{\mathrm{EG}}(z_{u,j}) = 2\lfloor \log_2(z_{u,j}+1) \rfloor + 1. \tag{5}$$

Since quantization and the floor operation are non-differentiable, we use the following smooth surrogate under the sign-balance assumption $p(r \geq 0 \mid q) = p(r < 0 \mid q) = \frac{1}{2}$:

$$\widetilde{\ell}_{\mathrm{EG}}(r_{u,j}) = 2\log_2\left(2^{b+1}|r_{u,j}| + \frac{3}{2}\right) + 1. \tag{6}$$

The residual rate of generalized layer u is the average code length over its N_u residual elements:

$$R_u = \frac{1}{N_u}\sum_{j=1}^{N_u} \widetilde{\ell}_{\mathrm{EG}}(r_{u,j}). \tag{7}$$

It measures the description cost of the layer's forward update: a low rate indicates a compressible update, whereas a high rate reflects a richer modification of the representation. Thus, R_u can be used to instantiate ψ to provide a differentiable signal of structural redundancy for structure learning.

3 Generalized Residual Constrained Training (GRCT)

3.1 Rate-Distortion Objective

The training of a learnable network can be viewed as a rate–distortion optimization:

$$\min_{\theta}\ D + \lambda R, \qquad \lambda \geq 0, \tag{8}$$

where D measures task distortion, R denotes structural description cost, and λ controls the trade-off between task fidelity and structural compression.

We integrate the soft gates defined in Section 2.1 into model training for learnable structural decisions. At a given structural granularity, the network is partitioned into non-overlapping generalized residual units $\mathcal{U} = \{1, \ldots, U\}$. For input x, unit u contributes the gated residual $g_u r_u(x)$, where $g_u = 1/(1+\exp(-s_u))$ is parameterized by a trainable logit $s_u \in \mathbb{R}$, enabling unconstrained and differentiable gate optimization while ensuring $g_u \in (0,1)$.

The unified GRCT objective can be expressed as:

$$\min_{\theta,\{s_u\}_{u=1}^{U}} \quad \mathcal{L}_{\mathrm{GRCT}} = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(f_{\theta,g}(x), y) + \frac{\lambda}{U}\sum_{u\in\mathcal{U}} g_u R_u\right], \tag{9}$$

where (x, y) denotes an input–target pair sampled from $\mathcal{D}$, while θ, f, and ℓ denote the model parameters, forward mapping, and loss function, respectively.

The first term in Equation 9 represents the task distortion $\mathcal{L}_{\mathrm{task}}$, which encourages the model to preserve the representational transformations required for the target task. The second term represents the structural rate $\mathcal{L}_{\mathrm{rate}}$, which is expressed as a gate-weighted average of the residual rates R_u introduced in Section 2.2. By weighting each residual rate by its corresponding gate, this term penalizes costly active units and reduces the model's overall structural cost. Model parameters and gates are jointly optimized under the competition of task performance and structural cost. During training, structural pruning is driven directly by $g_u R_u$, without requiring an extra importance score.

When the generalized layers correspond to Transformer layers, Attention and FFN sublayers, and multi-layer segments, we refer to the method as *GRCT*, *GRCT-SL*, and *GRCT-ML*, respectively.

3.2 Computational Optimization

GRCT achieves structural optimization through residual compression, which means different resource allocation and scheduling from the conventional training. Therefore, we introduce several computational optimizations to further reduce resource consumption during training.

Dynamic computation graph. Given a pruning threshold ε, when $g_u\, R_u < \varepsilon$, i.e., the gated residual output becomes negligible and stable. Retaining such a unit in subsequent forward and backward passes incurs unnecessary computation. Therefore, the unit is removed from the computation graph, for shorter propagation paths and lower accelerator memory usage.

Finite rate backpropagation. Each generalized residual represents the additive update contributed by its corresponding layer. Full backpropagation sends rate gradients through all preceding layers, lengthening the backward path and perturbing compression signals in shallow layers. We stop rate gradients at the input of each layer while preserving full task-loss backpropagation, thereby shortening the rate-gradient path, reducing cross-layer dependencies, and improving parallelism.

The optimized GRCT algorithm can be summarized as follows:

Algorithm 1 Generalized Residual Constrained Training (GRCT).

Require: Model parameters θ, network f and loss function ℓ; gate $\{g_u\}_{u=1}^{U}$, logits $\{s_u\}_{u=1}^{U}$ and generalized residual $\{r_u\}_{u=1}^{U}$; number of generalized layers U; training data $\mathcal{D}$; total training steps K; rate functions R; coefficient schedule λ; pruning threshold ε; learning rate η.

Ensure: A trained, deployable compact model.

1: Initialize $\mathcal{U} = \{1, \ldots, U\}$, $\mathcal{A} \leftarrow \mathcal{U}$, and $g_u = \sigma(s_u) \approx 1$ for every $u \in \mathcal{U}$.
2: **for** $k = 1, \ldots, K$ **do**
3: Sample a training pair $(x_k, y_k) \sim \mathcal{D}$.
4: Forward over active units $u \in \mathcal{A}$: $h_{u,\text{out}} = h_{u,\text{in}} + g_u\, r_u$.
5: Compute the training objective: $\mathcal{L} = \ell(f_{\theta,g}(x_k), y_k) + \dfrac{\lambda}{U} \displaystyle\sum_{u \in \mathcal{A}} g_u\, R_u$.
6: Update trainable parameters: $(\theta_u, s_u) \leftarrow (\theta_u, s_u) - \eta\, \nabla_{(\theta_u, s_u)} \mathcal{L}$.
7: Refresh the gates: $g_u = \sigma(s_u)$.
8: Remove newly pruned units: $\mathcal{A} \leftarrow \mathcal{A} \setminus \{u :\ g_u\, R_u < \varepsilon\}$.
9: **end for**
10: **return** the trained model with active structure $\mathcal{A}$.

3.3 Dynamic Structure Metrics

Metrics such as pruning ratio, memory usage, throughput, and speedup characterize system costs for a fixed model architecture, but do not capture GRCT's evolving computational graph. We introduce additional system-level metrics to characterize its dynamic structure.

Definition 1 (Structural exposure ratio). Suppose that the model contains layers $\{1, \cdots, U\}$ training for K steps. The active set at step k is $\mathcal{A}^{(k)} = \left\{u : g_u^{(k)}\, \psi\left(r_u^{(k)}\right) \geq \varepsilon\right\}$. $C_k(\cdot)$ measures the parameters in layers under the current step k. The structural exposure ratio is defined as

$$\rho_{\text{train}} = \frac{\sum_{k=1}^{K} C_k(\mathcal{A}^{(k)})}{\sum_{k=1}^{K} C_k(\{1, \ldots, U\})}. \tag{10}$$

ρ_{train} captures both how much structure is removed and when the removal occurs. Given the same final architecture, earlier structural contraction yields a lower ρ_{train}.

Definition 2 (Average training time and peak memory). Let $\mathcal{I}_k$ denote the time interval of optimization step k, $T(k)$ its wall-clock duration, and $m(\tau)$ the GPU memory usage at time τ. For a training run of K steps, the average step time and average peak GPU memory are defined as

$$\overline{T}_{\text{train}} = \frac{1}{K} \sum_{k=1}^{K} T(k), \qquad \overline{M}_{\text{train}} = \frac{1}{K} \sum_{k=1}^{K} \max_{\tau \in \mathcal{I}_k} m(\tau). \tag{11}$$

$\overline{T}_{\text{train}}$ and $\overline{M}_{\text{train}}$ characterize the average runtime and GPU memory demand throughout structural training, revealing the average time and memory costs across entire training trajectory.

These metrics provide a trajectory-aware evaluation of dynamic structure learning, distinguishing methods that achieve similar final architectures with different structural trajectories, and enabling a fairer assessment of their system resource usage during training.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models, Datasets and Baselines. We conduct extensive experiments on four models (LLaMA-3.2-1B (Grattafiori et al., 2024), Qwen3-4B (Qwen Team, 2025), LLaMA-3.1-8B (Grattafiori et al., 2024), and Qwen3-14B (Qwen Team, 2025)) and six tasks, including classification (BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2019), Devign (Zhou et al., 2019), and IMDB (Maas et al., 2011)) and generation (CodeAlpaca (Chaudhary, 2023) and SQuAD (Rajpurkar et al., 2016)). We adopt ShortGPT (Men et al., 2025), LaCo (Yang et al., 2024), LLM-Pruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024), and SkipGPT (Zhao et al., 2025) as external baselines for performance comparison, GRCT-SL as an internal baseline for algorithmic comparison, and Dense as the unpruned reference.

Evaluation Metrics. Model performance during training and inference is measured by accuracy, perplexity (PPL), and pruning rate of parameters. System metrics follow Sec. 3.3. During training, we report the structural exposure ratio ρ_{train}, the mean time $\overline{T}_{\text{train}}$ and memory $\overline{M}_{\text{train}}$, and the terminal time T_{end} and memory M_{end}. During inference, we report the parameter reduction ratio ρ_{param}, latency t_{inf}, throughput Φ_{inf}, speedup Sp_{inf}, and memory M_{inf}. These metrics are measured on RTX PRO 6000 GPUs. More details are provided in Appendix C.1.

4.2 MAIN RESULTS

We apply GRCT and GRCT-SL across models and datasets to evaluate their ability to adaptively learn architectures while preserving performance. The final pruning-rate trade-off of the models is summarized in Table 1. The end-to-end training and inference metrics are reported in Table 2.

Table 1: Structural pruning performance of GRCT and GRCT-SL against external baselines across models and tasks. Taking the better result between GRCT and GRCT-SL in each setting, accuracy decreases by only 0.40 percentage points on average relative to Dense across the 16 classification settings, while seven of the eight generation settings achieve lower PPL than Dense.

Model	Method	BoolQ		PIQA		Devign		IMDB		CodeAlpaca		SQuAD	
		Rate	Acc	Rate	Acc	Rate	Acc	Rate	Acc	Rate	PPL	Rate	PPL
LLaMA-3.2-1B	Dense	0.0%	79.24%	0.0%	73.78%	0.0%	67.17%	0.0%	95.38%	0.0%	3.12	0.0%	52.77
	ShortGPT	29.5%	37.80%	39.4%	49.56%	19.7%	53.81%	39.4%	53.75%	24.6%	633.04	24.6%	3.70×10^3
	LaCo	29.5%	37.80%	39.4%	49.67%	19.7%	56.26%	39.4%	56.34%	24.6%	439.19	24.6%	3.23×10^3
	LLM-Pruner	31.0%	64.31%	40.0%	62.24%	18.0%	64.60%	40.0%	92.82%	24.0%	3.25	24.0%	54.72
	SliceGPT	29.5%	58.29%	39.4%	52.18%	19.7%	62.55%	39.4%	80.32%	24.6%	23.34	24.6%	1.74×10^3
	SkipGPT	28.1%	66.64%	39.7%	50.22%	20.5%	61.46%	34.8%	93.53%	24.5%	4.15	24.6%	39.39
	GRCT-SL	34.5%	80.09%	**43.5%**	**73.72%**	22.2%	68.08%	44.3%	93.56%	35.3%	3.41	25.6%	116.43
	GRCT	**39.4%**	**79.85%**	44.3%	72.74%	**24.6%**	**67.20%**	**49.2%**	**93.90%**	**24.6%**	**2.44**	**24.6%**	**37.07**
Qwen3-4B	Dense	0.0%	86.76%	0.0%	85.75%	0.0%	68.23%	0.0%	95.74%	0.0%	2.97	0.0%	45.97
	ShortGPT	39.7%	56.06%	42.2%	49.51%	19.8%	55.71%	49.7%	50.04%	42.2%	6.59×10^4	27.4%	6.98×10^3
	LaCo	39.7%	47.68%	42.2%	49.29%	19.8%	52.34%	49.7%	50.00%	42.2%	1.40×10^5	27.4%	4.38×10^3
	LLM-Pruner	38.3%	69.42%	39.2%	60.45%	19.1%	64.31%	47.8%	91.15%	39.2%	3.14	26.7%	25.97
	SliceGPT	39.8%	62.17%	42.2%	49.95%	19.8%	55.31%	49.7%	57.23%	42.2%	3.16×10^3	27.4%	1.88×10^3
	SkipGPT	40.5%	61.53%	42.1%	49.40%	20.3%	58.93%	48.9%	90.94%	42.1%	3.08	27.1%	25.20
	GRCT-SL	41.6%	86.54%	**47.2%**	**84.00%**	21.3%	66.95%	**67.0%**	**93.34%**	50.2%	3.11	43.5%	90.13
	GRCT	**42.2%**	**87.03%**	44.7%	81.28%	**34.8%**	**69.07%**	52.1%	94.86%	**42.2%**	**2.21**	**27.4%**	**20.11**
LLaMA-3.1-8B	Dense	0.0%	88.62%	0.0%	78.78%	0.0%	64.17%	0.0%	95.37%	0.0%	3.04	0.0%	64.24
	ShortGPT	51.6%	40.73%	54.3%	49.46%	73.4%	59.77%	59.8%	50.11%	32.6%	513.88	27.2%	1.30×10^3
	LaCo	51.6%	52.81%	54.3%	50.38%	73.4%	64.13%	59.8%	50.00%	32.6%	7.75×10^3	27.2%	1.26×10^3
	LLM-Pruner	51.1%	72.54%	54.8%	66.49%	54.8%	64.17%	54.8%	93.02%	28.3%	2.66	27.1%	62.72
	SliceGPT	51.6%	62.17%	55.4%	51.03%	73.4%	60.29%	59.8%	53.94%	32.6%	15.13	27.2%	483.10
	SkipGPT	50.3%	84.34%	54.1%	74.27%	73.3%	64.28%	59.5%	91.04%	32.1%	2.89	26.6%	52.70
	GRCT-SL	**53.8%**	**86.27%**	**55.4%**	**82.64%**	73.4%	65.01%	60.8%	93.81%	33.5%	3.01	29.0%	71.05
	GRCT	51.6%	85.35%	59.8%	75.14%	**81.5%**	**65.67%**	**65.2%**	**94.10%**	**32.6%**	**2.62**	**27.2%**	**51.10**
Qwen3-14B	Dense	0.0%	89.76%	0.0%	89.88%	0.0%	66.25%	0.0%	96.34%	0.0%	2.80	0.0%	29.02
	ShortGPT	44.7%	59.39%	38.0%	49.73%	31.3%	43.45%	55.9%	51.58%	51.4%	2.82×10^5	53.7%	1.23×10^6
	LaCo	44.7%	44.74%	38.0%	49.51%	31.3%	43.45%	55.9%	49.60%	51.4%	8.05×10^4	53.7%	1.80×10^6
	LLM-Pruner	38.0%	77.65%	37.5%	76.88%	30.1%	65.96%	55.4%	93.02%	48.6%	3.46	53.1%	39.90
	SliceGPT	44.7%	62.17%	37.9%	50.65%	31.8%	44.25%	55.9%	54.13%	51.4%	1.16×10^3	53.7%	5.93×10^3
	SkipGPT	43.7%	62.17%	37.8%	49.67%	31.7%	60.80%	54.9%	90.73%	51.4%	2.92	53.2%	46.34
	GRCT-SL	56.4%	84.77%	**38.0%**	**87.54%**	**31.8%**	**67.46%**	**70.8%**	**94.15%**	63.3%	4.30	66.0%	222.37
	GRCT	**44.7%**	**86.06%**	47.0%	83.51%	40.3%	65.67%	55.9%	94.30%	**51.4%**	**2.27**	**53.7%**	**37.86**

Comparison with pruning baselines. GRCT generally outperforms pruning baselines at comparable or higher pruning rates, even matching or surpassing the unpruned model. It achieves up to 93.8% of layer pruning (81.5% of parameters). GRCT-SL shows degradation on generation tasks, which is discussed in Section 5. At comparable pruning rates, ShortGPT, LaCo, and SliceGPT without recovery degrade sharply, while LLM-Pruner and SkipGPT still trail GRCT broadly despite recovery fine-tuning. These results suggest that GRCT's autonomous structure learning achieves a better balance between compression and task performance than the pruning-then-recovery paradigm.

Performance across models and tasks. The structures learned by the GRCTs exhibit clear scale- and task-dependent patterns. Larger models typically tend to support higher pruning rates, suggesting more removable structural redundancy. More complex tasks generally require more structural depth to be retained. These findings demonstrate that GRCTs can adaptively learn structures tailored to model scale and task requirements without a fixed compression target.

Table 2: End-to-end system efficiency of GRCT and GRCT-SL relative to Dense across models trained on IMDB as an example. Training time is reported in ms/step, memory in GiB, inference latency in ms/batch, and throughput in samples/s.

Model	Method	Training dynamics					Final structure and inference				
		ρ_{train}	$\overline{T}_{\text{train}}$	$\overline{M}_{\text{train}}$	T_{end}	M_{end}	ρ_{param}	t_{inf}	Φ_{inf}	Sp_{inf}	M_{inf}
LLaMA-3.2-1B	Dense	100%	865	6.61	865	6.61	0.0%	160.0	200.01	1.00×	3.81
	GRCT-SL	58.8%	834	5.42	435	4.57	44.3%	76.5	418.18	2.09×	2.29
	GRCT	54.4%	716	5.52	348	4.40	49.2%	57.5	556.13	2.78×	2.18
Qwen3-4B	Dense	100%	3532	17.96	3532	17.96	0.0%	638.8	50.10	1.00×	11.01
	GRCT-SL	47.9%	3011	12.12	1205	8.47	66.4%	246.7	129.73	2.59×	3.74
	GRCT	53.8%	2810	13.15	1409	9.69	55.2%	248.8	128.61	2.57×	4.59
LLaMA-3.1-8B	Dense	100%	5408	31.56	5408	31.56	0.0%	793.5	40.33	1.00×	17.80
	GRCT-SL	50.6%	4604	20.68	2116	15.15	62.1%	257.5	124.30	3.08×	7.11
	GRCT	42.5%	3530	18.60	1368	12.06	69.7%	195.2	163.91	4.06×	6.05
Qwen3-14B	Dense	100%	10414	56.91	10414	56.91	0.0%	1946.8	16.44	1.00×	30.79
	GRCT-SL	47.5%	8517	35.38	3648	24.19	68.9%	573.0	55.84	3.40×	10.33
	GRCT	49.4%	7549	35.17	3297	23.69	63.8%	579.9	55.18	3.36×	11.67

System efficiency. GRCT variants progressively contract model structure, yielding system benefits in both training and inference. In Table 2, the GRCT variants expose only 42.5-58.8% of the structure over the full training trajectory with less average time and memory. By the end of training, they achieve up to a $3.95\times$ speedup and 61.8% memory savings. In inference, GRCT delivers up to a $4.06\times$ speedup and 66.0% memory savings. Moreover, these gains generally become more pronounced as model scale increases, which establish GRCT as an efficiency-aware approach to structure learning.

4.3 Training Trade-off

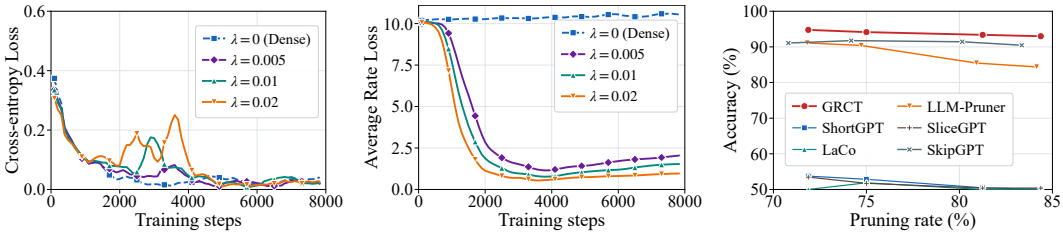
GRCT balances task performance and architectural complexity throughout training, making it essential to analyze the performance of its trade-off. We vary the Lagrange multiplier λ to control this balance and analyze how it manifests in the training trajectory and the final model.

Figs. 2a and 2b show how the task loss and the structural rate evolve under different λ. The loss undergoes decline, oscillation, and convergence, whereas the rate drops rapidly before gradually converging. This indicates the detailed learning process of GRCT: the model first compresses the removable redundancy; as compression approaches its limit, the competition between task performance and the rate constraint drives it to search for a better architecture; then it eventually converges.

Fig. 2c compares the accuracy–pruning-rate trade-off of the final GRCT models to the baselines. As the pruning rate increases, GRCT's accuracy declines only slowly, whereas the baselines degrade markedly in the high-pruning-rate regime. This shows that GRCT maintains task performance at high compression rates and, in turn, achieves more stable structural optimization.

5 Discussion

Evolution of Residual Distribution Building on the GG observation in Section 2.2, GRCT applies an EG rate constraint to minimize residual coding cost during training. To investigate how



(a) Cross-entropy loss of tasks. (b) Average EG coding rate. (c) Pruning-accuracy trade-off.

Figure 2: Training trade-off and dynamics on IMDB with LLaMA-3.1-8B under different λ versus various pruning baselines, where $\lambda = 0$ corresponds to the Dense model.

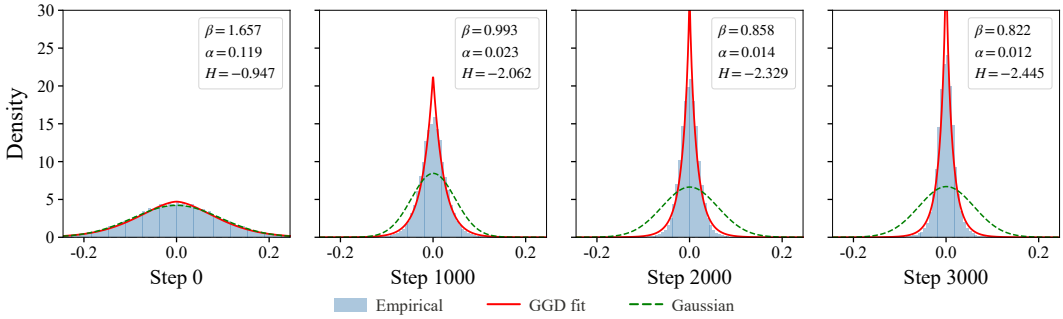
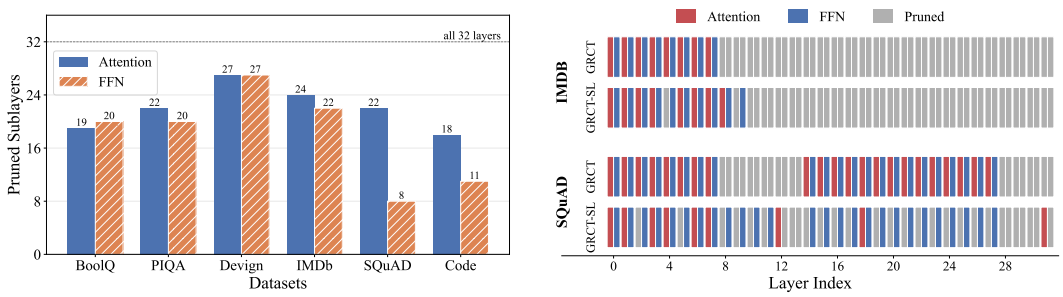


Figure 3: Evolution of residual distribution at decoder layer 16 during GRCT training of LLaMA-3.1-8B on IMDB fitted by GGD and Gaussian distributions. Every stage reports the fitted GGD shape β, scale α, and differential entropy H.



(a) Number of Attention and FFN sublayers pruned by GRCT-SL in the runs reported in Table 1.

(b) Per-layer pruning behavior of GRCT and GRCT-SL in IMDB and SQuAD tasks.

Figure 4: Sublayer pruning behavior of GRCT and GRCT-SL with LLaMA-3.1-8B across tasks.

the rate constraint reshapes the residual distribution, we further track this evolution by comparing residual distributions at different training stages in Fig. 3. Across all stages, the empirical residual distributions are consistently well fitted by GGDs. As training progresses, the fitted distributions become more sharply peaked and heavy-tailed. Moreover, their differential entropy steadily decreases, which is consistent with the reduction in the average EG rate in Fig. 2b. These results indicate that EG rate optimization guides the model to reshape residual statistics into more compact, low-entropy distributional representations, embodying GRCT's compression-as-pruning characteristic.

Distribution of Pruned Layers. Finer granularity typically implies more refined structural pruning. But in Table 1, GRCT-SL underperforms GRCT on the generation tasks. We further examine the pruning distributions of GRCT and GRCT-SL to understand this phenomenon. Fig. 4a shows that GRCT-SL prunes similar numbers of Attention and FFN sublayers for classification, but more Attention sublayers for generation. One possible explanation is that teacher forcing allows few Attention sublayers to mediate cross-token information exchange during training, while FFNs provide most of the token-level transformations and GRCT-SL reports loss of only 0.105 in SQuAD training. But excessive Attention pruning consequently weakens contextual modeling on test examples

with loss of 4.263. By pruning Attention and FFN together, GRCT avoids this imbalance and better preserves generation performance. Fig. 4b shows that classification pruning concentrates in later layers, suggesting that earlier representations capture much of the needed information. Generation pruning is spread across depth, demonstrating more distributed inter-layer dependencies.

6 Related Work

Structural Pruning Structured pruning reduces model cost by removing complete computational units. Early work selects among attention heads, FFN dimensions, or coupled parameter groups; for example, LLM-Pruner (Ma et al., 2023) estimates the importance of coupled structures from gradient information. SNIP (Lin et al., 2020) uses activation thresholding to identify redundant modules, followed by iterative pruning and retraining. CoFi (Xia et al., 2022) first completes target-task fine-tuning of a teacher model, then guides student pruning with a prescribed sparsity target and performs long-term recovery fine-tuning. Post-training layer pruning adopts whole Transformer blocks as compression units: Shortened LLaMA (Kim et al., 2024) measures the damage caused by deletion using the change in perplexity or a Taylor criterion, ShortGPT (Men et al., 2025) defines Block Influence from the similarity between a layer's input and output, and Gromov et al. (2025) remove consecutive deep layers according to cross-layer representation similarity; LaCo (Yang et al., 2024) retains information from compressed layers by merging rather than discarding them. Recent methods further relax pruning granularity: FinerCut (Zhang et al., 2024) and BlockPruner (Zhong et al., 2025) treat Attention and FFN as independent candidates and drive the search using output perturbation or perplexity, while TaBP (Jo et al., 2026) estimates block importance from task-conditioned output distributions. Most of these post-training methods compute fixed calibration metrics on a trained model before ranking, greedy search, distillation, or merging. In contrast, GRCT treats layer importance as a cost compressed during optimization jointly by task distortion and structural rate, directly yielding a compact model from the end-to-end-training.

Variable-Depth Transformers Several works also improve the adaptability of models to different execution depths during training. LayerDrop (Fan et al., 2019) trains multi-depth subnetworks with shared parameters via whole-layer structured dropout, and DynaBERT (Hou et al., 2020) distills a model whose width and depth can be selected according to the deployment budget. Early-exit and conditional-computation methods instead adapt the actual execution path to the input: CALM (Schuster et al., 2022) extends early exit to autoregressive generation, LayerSkip (Elhoushi et al., 2024) combines layer dropout, early-exit supervision, and self-speculative decoding, Mixture-of-Depths (Raposo et al., 2024) allocates different depths to different tokens under a fixed compute budget, and IG-Pruning (Qiao et al., 2025) selects block masks per input at inference. These methods typically prescribe candidate depths or compute budgets in advance and use layer skipping to improve model robustness or shorten the computation path, while generally retaining the full parameter structure. In contrast, GRCT progressively simplifies the model during structure learning, contracting both the effective training computation graph and the model architecture, and learns the final network through joint task–structure optimization.

7 Conclusion

We propose GRCT, a rate–distortion paradigm that learns Transformer structure during training. Motivated by the GG observations, GRCT models EG coding rates of generalized residuals as structural costs and incorporates them into the objective, thereby jointly optimizing task performance and structural redundancy during training and directly yielding a deployable compact model without a separate post-training pruning or recovery stage. Across diverse models and tasks, GRCT consistently achieves better performance–pruning balances than all the pruning baselines. At a layer pruning rate of 93.8%, GRCT even improves accuracy by 1.5 percentage points over the unpruned Dense model. During the end-to-end training and deployment, GRCT reduces training time and memory by up to 34.7% and 41.1%, while achieving up to $4.06\times$ speedup and 66.0% memory savings in inference. GRCT unifies task learning and structure optimization in a single training run, transforming compression from preset post-training pruning into autonomous structure learning. It further extends the focus of model compression from the final model to the entire training trajectory, inspiring a training-aware and structure-adaptive perspective on model design and optimization.

AI USE STATEMENT

Generative AI tools assisted with literature discovery, language polishing, and preparation of the GRCT-ML appendix visualization. They were not used to generate experimental data, develop theoretical models or mathematical claims, formulate proofs or hypotheses, design experiments or research methods, or implement the training methods. The authors checked the plotted values and structural masks against the evaluation records and take responsibility for the cited references and all content and claims in this work.

REFERENCES

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2401.15024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The unreasonable ineffectiveness of the deeper layers. In *International Conference on Learning Representations*, volume 2025, pp. 81906–81920, 2025.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33: 9782–9793, 2020.

Song-ha Jo, Youngrok Ko, Sang-goo Lee, and Jinseok Seol. Task-aware block pruning with output distribution signals for large language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6089–6107, 2026.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. Shortened llama: Depth pruning for large language models with comparison of retraining methods. *arXiv preprint arXiv:2402.02834*, 2024.

Zi Lin, Jeremiah Liu, Zi Yang, Nan Hua, and Dan Roth. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 719–730, 2020.

Shizhao Liu, Anton Pletenev, Ralf M Haefner, and Adam C Snyder. Task learning increases information redundancy of neural responses in macaque visual cortex. *Science*, 391(6789):1029–1035, 2026.

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

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 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.

Kangyu Qiao, Shaolei Zhang, and Yang Feng. Ig-pruning: Input-guided block pruning for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10629–10640, 2025.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*, 2024.

Mototaka Suzuki, Cyriel MA Pennartz, and Jaan Aru. How deep is the brain? the shallow brain hypothesis. *Nature Reviews Neuroscience*, 24(12):778–791, 2023.

Jiangtao Wen and John D. Villasenor. Structured prefix codes for quantized low-shape-parameter generalized gaussian sources. *IEEE Transactions on Information Theory*, 45(4):1307–1314, 1999. doi: 10.1109/18.761289.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1513–1528, 2022.

Yifei Yang, Zouying Cao, and Hai Zhao. Laco: Large language model pruning via layer collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6401–6417, 2024.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. FinerCut: Finer-grained interpretable layer pruning for large language models. *arXiv preprint arXiv:2405.18218*, 2024. doi: 10.48550/arXiv.2405.18218. URL https://arxiv.org/abs/2405.18218.

Anhao Zhao, Fanghua Ye, Yingqi Fan, Junlong Tong, Zhiwei Fei, Hui Su, and Xiaoyu Shen. Skipgpt: Dynamic layer pruning reinvented with token awareness and module decoupling. *arXiv preprint arXiv:2506.04179*, 2025.

Longguang Zhong, Fanqi Wan, Ruijun Chen, Xiaojun Quan, and Liangzhi Li. Blockpruner: Fine-grained pruning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5065–5080, 2025.

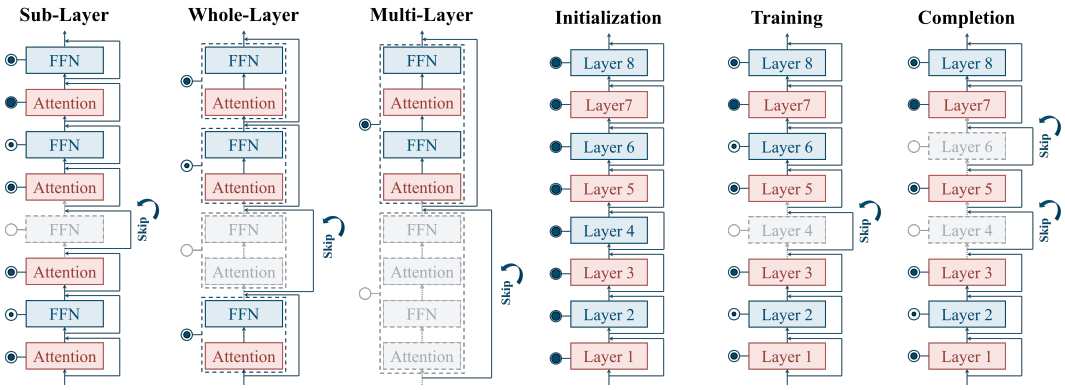
Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.

A Overview of GRCT Algorithm

Figure 5 provides an overview of the GRCT algorithm from two perspectives: the structural granularities (Figure 5a) and its actual learning process (Figure 5b).

Figure 5a illustrates structural learning at different granularities. Each granularity uses one gate per generalized layer. When the contribution of a generalized layer becomes sufficiently tiny, the hidden states bypass the pruned structure directly through the surrounding skip connections. At the layer granularity, GRCT removes entire Transformer layers; at the sublayer granularity, it removes individual Attention or FFN sublayers within a layer, so the two sublayers of the same layer can be pruned independently.

Figure 5b illustrates the actual learning process of GRCT. It starts from the dense model, where all gates are open and all layers participate in the forward computation. Once the model learns that the contributions of certain units are sufficiently small and stable, these units are removed from the computation graph and offloaded from GPU memory. The active structure contracts progressively until the end of training, where the remaining structure is exported directly as a deployable compact model, without any extra pruning or recovery stage.



(a) Structural learning at different granularities. (b) The actual learning process of GRCT.

Figure 5: Overview of the GRCT algorithm.

B GG Observation of Residuals Across Granularities

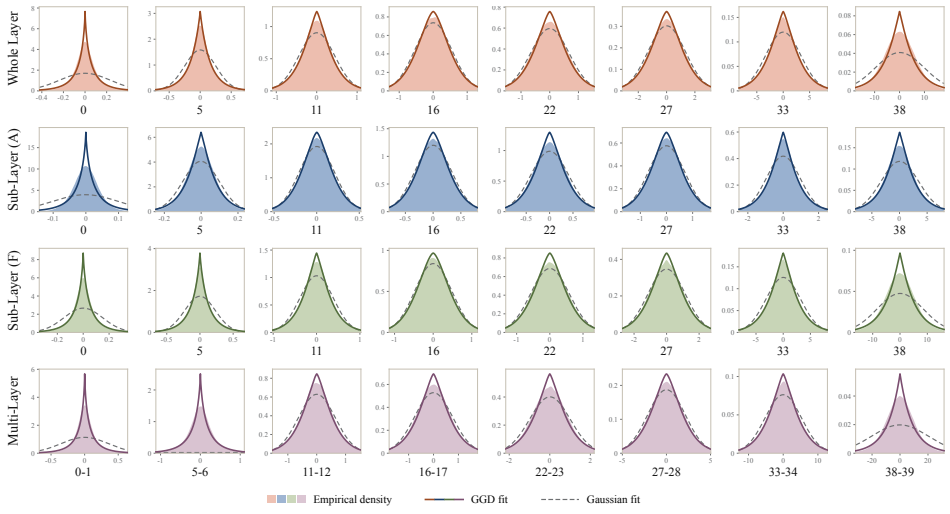


Figure 6: Empirical residual densities with fitted GGD and Gaussian distributions on Qwen3-14B / Devign, at granularities of whole layer, Attention sublayer, FFN sublayer, and two-layer segment.

Section 2.2 examines the residual statistics of individual Transformer layers on DeepSeek-R1-Distill-Qwen-7B / SQuAD in Figure 1. We further verify the GG observation across different layer granularities in Figure 6, taking the training of Qwen3-14B on the Devign dataset as an instance.

Figure 6 presents the GG distributions of the generalized residuals output by four types of structural units: whole Transformer layers, Attention sublayers, FFN sublayers, and contiguous two-layer segments, across various layers. At every granularity, the residuals closely follow the GGD in contrast to the Gaussian fit, which systematically produces overly flat peaks and overly heavy tails, and the fitted shape parameters remain well below 2 ($\beta \in [0.69, 1.58]$). This corroborates that GRCT-SL and GRCT-ML can be designed and trained in the same manner as GRCT. Moreover, although the distributions differ slightly across granularities, they are largely homogeneous and isomorphic in space, indicating that the residual signal exhibits stable statistical characteristics across different spatial layouts of layers.

C Additional Experiments

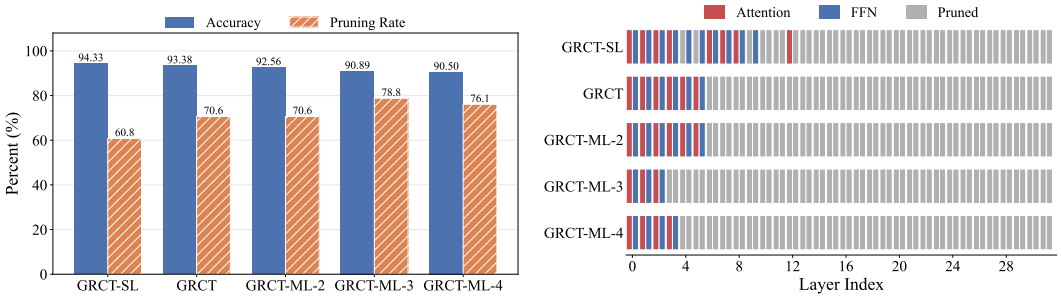
C.1 Implementation Details

Training. All tasks use full-parameter fine-tuning with a learning rate 2×10^{-5}, batch size 16, and the AdamW optimizer (β_1=0.9, β_2=0.999); the gate logits use a separate learning rate of 5×10^{-3} in GRCT. Classification and generation tasks train for 10 and 5 epochs, respectively, for all GRCT variants and baselines. The LLM-Pruner and SkipGPT baselines use additional recovery fine-tuning following their official settings.

Hyperparameters. The rate is computed with b=8 quantization bits in Equation 6, and gates are initialized to $g_u = 0.99$. The pruning threshold is set to $\varepsilon = 1$ and the seed to 42. The Lagrange multiplier is set as $\lambda = 0.02$ for classification and $\lambda = 0.2$ for generation. In the trade-off experiments, λ varies over $\{0.005, 0.01, 0.02, 0.05\}$ for classification and $\{0.05, 0.1, 0.2, 0.5\}$ for generation. All sequences use a maximum length of 512 tokens.

C.2 Ablation on Structural Granularity

GRCT can be applied to hierarchical pruning at different granularities. In Section 4, we presented the training results of GRCT and GRCT-SL. Here we further investigate GRCT-ML, which applies the generalized residual constraint to contiguous multi-layer segments. The L Transformer layers are partitioned into non-overlapping segments of k consecutive layers, and each segment shares a single gate and rate; the variants spanning k=2, 3, and 4 layers are denoted GRCT-ML-2, GRCT-ML-3, and GRCT-ML-4, respectively. We train them on LLaMA-3.1-8B / IMDB (16 segments of 2 layers, 11 segments of at most 3 layers, and 8 segments of 4 layers, respectively), using the same settings as GRCT. The experimental results are shown in Figure 7.



(a) Test accuracy and full-model parameter pruning rate.

(b) Distributions of the layer pruning.

Figure 7: Effect of structural granularity on LLaMA-3.1-8B / IMDB, comparing accuracy and parameter pruning rate (removed parameters divided by those of the original 8B model) together with the final layer-wise structures.

Figure 7a shows a broadly similar accuracy–compression trade-off across pruning granularities, suggesting that the learned structure is shaped primarily by the balance between task distortion and structural rate rather than by granularity. Coarser pruning units tend to increase the pruning rate at some cost in accuracy. As all layers within a segment share one gate, a larger segment offers fewer possible pruning boundaries and may force useful layers to be removed together with redundant ones. Figure 7b shows a largely consistent depthwise pattern: retained computation is concentrated in early layers, while pruning occurs mainly in later layers. Thus, the broad location of learnable redundancy appears stable across granularities, even though the exact boundary and resulting trade-off vary with the pruning unit.