LLARA: LOW-RANK LINEAR ATTENTION RESIDUAL ADAPTATION FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern large language models (LLMs) rely critically on residual connections with PreNorm, yet their uniform aggregation of layer outputs with fixed unit weights causes hidden states to expand uncontrollably with depth, gradually diluting the contribution of each layer. Recent research introduces Attention Residuals (AttnRes) to enable each layer to selectively aggregate earlier representations on a per-token basis through learned weights, by replacing the fixed accumulation with softmax attention over preceding layer outputs, yielding consistent empirical gains across model scales. However, pre-training AttnRes-based LLMs demands prohibitively high computational resources and expenses, preventing a wide range of open-source models from reaping the benefits of this novel architecture. To bridge this gap, we propose **L**ow-Rank **L**inear **A**ttention **R**esidual **A**daptation, or **LLARA**, which freezes the pre-trained model weights and introduces a learnable low-rank linear attention residual mechanism to fine-tune depth-wise aggregation of per-block residual contributions, allowing pre-trained LLMs to robustly benefit from token-wise attention residuals during downstream task adaptation. Jointly with LoRA for parameter-efficient fine-tuning, LLARA maintains a compact linear attention state along the decoder depth through low-rank key–value trajectories, reads it out at the final block via a learnable query, and adds the resulting residual back to the hidden representations, thereby fine-tuning the model's residual architecture. Both theoretical analysis and empirical experiments have demonstrated the effectiveness, efficiency, and robustness of LLARA. Our code is available at `https://anonymous.4open.science/r/LLARA`.

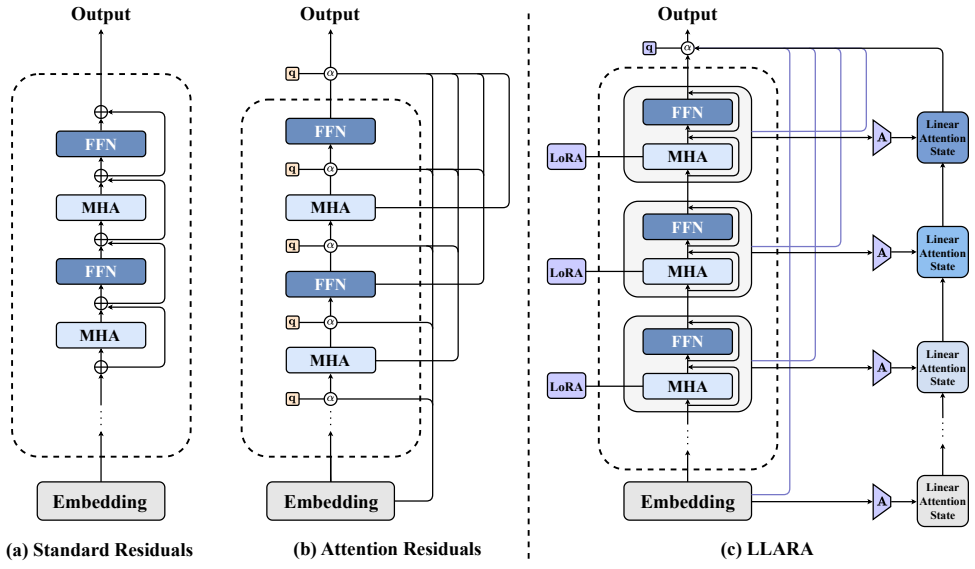


Figure 1: Overview of LLARA. **(a)** Standard Residuals: standard residual connections with uniform additive accumulation. **(b)** Attention Residuals: each layer selectively aggregates all previous layer outputs via learned attention weights. **(c)** LLARA: a learnable low-rank linear attention residual mechanism to fine-tune depth-wise aggregation of per-block residual contributions while freezing the pre-trained LLMs. Purple marks trainable parameters and residual links of LLARA.

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.

1 INTRODUCTION

Residual connections (He et al., 2016) with PreNorm (Xiong et al., 2020) are foundational to modern large language models (LLMs) (OpenAI, 2023; Llama Team, 2024; Qwen Team, 2025; DeepSeek-AI, 2025), acting as gradient highways that enable stable optimization at scale. However, the standard formulation $\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_l(\boldsymbol{h}_{l-1})$ follows a uniform, unweighted accumulation of all preceding layer outputs. Unrolling the recurrence gives $\boldsymbol{h}_l = \boldsymbol{h}_0 + \sum_{i=1}^{l} f_i(\boldsymbol{h}_{i-1})$, which expresses $\boldsymbol{h}_l$ as an initial representation plus the cumulative sum of all preceding residual increments. Let $\Delta_i = f_i(\boldsymbol{h}_{i-1})$ denote the residual increment at layer i. The stabilizing effect of PreNorm can be captured by depth-stable residual increments, i.e., for constants $0 < c \leq C$ independent of l, $c\sigma^2 \leq \mathbb{E}\|\Delta_i\|_2^2 \leq C\sigma^2$ for all $i \leq l$. Moreover, in high-dimensional hidden spaces, different residual increments are weakly aligned, i.e., for some constant $\eta < c$, $\left|2\sum_{1\leq i<j\leq l} \mathbb{E}\langle \Delta_i, \Delta_j\rangle\right| \leq \eta l\sigma^2$. Then $\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 = \sum_{i=1}^{l} \mathbb{E}\|\Delta_i\|_2^2 + 2\sum_{i<j} \mathbb{E}\langle \Delta_i, \Delta_j\rangle = \Theta(l\sigma^2)$, which makes the residual stream behave like a depth-wise random walk. Consequently, the relative contribution of any single layer is $O(1/\sqrt{l})$, linking uniform residual accumulation to progressive depth-wise dilution. As a result, this dilution inherently suppresses early signals and deeper layers are forced to produce ever-larger outputs to remain visible (Li et al., 2026). Meanwhile, recent studies (Gromov et al., 2025; Men et al., 2025; Csordás et al., 2025) suggest that not all layers are equally useful, yet the prevailing residual paradigms (Srivastava et al., 2015; Bachlechner et al., 2020; Wang et al., 2022; Pagliardini et al., 2024; Zhu et al., 2025; Xie et al., 2025; Zhang et al., 2026; Li et al., 2026) provide no per-token mechanism for a layer to selectively amplify or suppress earlier signals.

To address this issue, the recent Attention Residuals (AttnRes) (Kimi Team, 2026a;b) replaces the fixed additive recurrence with a learned softmax attention over depth. By equipping each layer with a learnable pseudo-query, AttnRes allows a model to compute input-dependent weights over all previous layer outputs, transforming depth-wise aggregation from a static sum into a content-aware retrieval process. This yields more balanced gradient distributions, bounded hidden-state growth, and consistent empirical gains across model sizes (Kimi Team, 2026a). However, realizing these benefits requires LLMs to be pre-trained from scratch with AttnRes integrated, which demands prohibitively high computational resources and expenses. For the vast majority of researchers and practitioners who fine-tune existing pre-trained LLMs rather than pre-training their own, the architectural advantages of attention residuals remain entirely inaccessible, creating a substantial gap between this promising mechanism and its real-world deployment.

In this paper, we propose a novel model architecture fine-tuning paradigm that introduces the attention residual mechanism into pre-trained LLMs in a parameter-efficient manner during post-training. Specifically, we introduce **L**ow-Rank **L**inear **A**ttention **R**esidual **A**daptation, or **LLARA**, a post-training recipe that freezes the pre-trained LLM while augmenting its forward computation with a compact linear-attention residual pathway. Rather than instantiating full-depth AttnRes caches, LLARA threads a compact linear attention state along decoder depth via low-rank key-value trajectories over per-block residual increments, updating fixed-size online accumulators as a surrogate for depth-wise softmax mixing. Only at the last decoder block, a small learnable query performs a single readout from this state and adds the resulting residual back into the hidden representations, recovering token-wise, input-dependent control over how residual increments are composed across depth. Jointly with LoRA (Hu et al., 2022) for parameter-efficient fine-tuning, LLARA learns low-rank adapters in the frozen backbone together with an input-dependent depth-wise residual readout, enabling attention-residual-style depth mixing when post-training standard pre-trained LLMs. Experiments confirm that LLARA consistently improves over a matched LoRA baseline across knowledge, mathematical reasoning, and code generation under both supervised and reinforcement post-training. The gains transfer across PreNorm residual backbones and increase with scale from 1.5B to 32B, indicating that uniform residual aggregation leaves a complementary depth-mixing degree of freedom unoccupied by within-block adapters. The contributions are summarized as follows:

- **Analytical Theoretical Guidance.** We characterize depth-wise dilution under uniform residuals and the role of normalized depth attention in selective aggregation, providing analytical guidance that informs post-training improvements to attention residual design.
- **Novel Adaptation Paradigm.** We introduce LLARA, a parameter-efficient post-training recipe that freezes the pretrained backbone and learns token-dependent cross-block residual composition via low-rank linear attention with a single output readout, compatible with LoRA.

- **In-depth Experimental Analysis.** Through extensive experiments, we verify that in-place AttnRes is structurally mismatched to pretrained residual recurrences, whereas LLARA trains stably and learns token-wise depth aggregation whose selected layers differ across domains, consistent with our gradient-stability and selective-aggregation analysis.

2 PRELIMINARIES

We consider a decoder-only Transformer (Radford et al., 2018) with L layers and hidden width d. For clarity, we describe the computation for a single token, omitting the batch and sequence dimensions. Let $\boldsymbol{h}_0 \in \mathbb{R}^d$ denote the token embedding or initial hidden representation, and for $l = 1, \ldots, L$, let $\boldsymbol{h}_l \in \mathbb{R}^d$ denote the hidden state after layer l. Residual learning (He et al., 2016) is critical for training deep networks, as it allows information and gradients to bypass layer transformations through identity mappings. Specifically, each layer updates the hidden state as:

$$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_l(\boldsymbol{h}_{l-1}), \tag{1}$$

where f_l represents the transformation applied by layer l, with each multi-head self-attention or feed-forward sublayer in LLMs counted as an individual layer.

Residual streams as uniform depth aggregation. Eq. (1) reveals that a residual stream is a fixed aggregation process along the depth dimension. Define the depth residual values as:

$$\boldsymbol{v}_0 = \boldsymbol{h}_0, \quad \boldsymbol{v}_l = \boldsymbol{h}_l - \boldsymbol{h}_{l-1} = f_l(\boldsymbol{h}_{l-1}), \quad l = 1, \ldots, L. \tag{2}$$

Unrolling Eq. (1) yields:

$$\boldsymbol{h}_l = \sum_{i=0}^{l} \boldsymbol{v}_i = \boldsymbol{h}_0 + \sum_{j=1}^{l} f_j(\boldsymbol{h}_{j-1}). \tag{3}$$

Thus, the standard residual connection assigns every realized residual increment a fixed unit coefficient. Although this identity path is essential for optimization, each layer nevertheless conditions only on the aggregated hidden state $\boldsymbol{h}_{l-1} = \sum_{i=0}^{l-1} \boldsymbol{v}_i$ rather than on the preceding increments $\{\boldsymbol{v}_i\}_{i<l}$ as separate quantities, so it lacks an explicit mechanism to selectively amplify or suppress particular earlier residual contributions along depth.

Attention residuals over depth. Attention Residuals (AttnRes) (Kimi Team, 2026a) generalizes the fixed summation in Eq. (3) by replacing uniform coefficients with learned input-dependent attention weights over depth. AttnRes sets $\boldsymbol{v}_0 = \boldsymbol{k}_0 = \boldsymbol{h}_0$ and $\boldsymbol{v}_i = \boldsymbol{k}_i = f_i(\boldsymbol{h}_{i-1})$ for $i \geq 1$, and forms the hidden state $\boldsymbol{h}_l$ by applying softmax attention to residual values up to depth l:

$$\boldsymbol{h}_l = \sum_{i=0}^{l} \alpha_{i\to l} \boldsymbol{v}_i = \alpha_{0\to l} \boldsymbol{h}_0 + \sum_{j=1}^{l} \alpha_{j\to l} f_j(\boldsymbol{h}_{j-1}), \tag{4}$$

$$\alpha_{i\to l} = \frac{\exp\left(\boldsymbol{q}_l^\top \mathrm{RMSNorm}(\boldsymbol{k}_i)\right)}{\sum_{j=0}^{l} \exp\left(\boldsymbol{q}_l^\top \mathrm{RMSNorm}(\boldsymbol{k}_j)\right)}, \tag{5}$$

where $\boldsymbol{q}_l \in \mathbb{R}^d$ is a layer-specific learnable pseudo-query initialized to $\mathbf{0}$ for training stability (so that depth weights start uniform), and the RMSNorm (Zhang & Sennrich, 2019) inside the attention kernel prevents large-magnitude residual values from dominating the depth weights. Thus, standard residual connections are the equal-weight special case of depth aggregation, whereas AttnRes enables each layer to retrieve earlier residual increments with learned, normalized, token-wise weights.

3 LLARA: LOW-RANK LINEAR ATTENTION RESIDUAL ADAPTATION

The theoretical analysis in Appendix C identifies depth-wise attention over residual increments as a principled remedy for the depth-wise dilution induced by uniform residual accumulation. Realizing this mechanism inside an existing pre-trained LLM during post-training introduces three design constraints. **First**, the depth axis should be formulated at the block level, consistent with the structure around which pretrained parameters and modern PEFT infrastructure are organized.

Second, the standard residual recurrence and the block input interface should not be rewritten mid-network, since pretrained weights are optimized under Eq. (1) and replacing intermediate states with an AttnRes-style depth aggregator shifts the input distribution of downstream blocks and can cause functional collapse. **Third**, the added depth-wise attention path should remain parameter-efficient and avoid materializing a full depth-wise activation cache, so that it remains compatible with activation recomputation and does not substantially increase GPU memory during post-training. LLARA is organized around these constraints through three central design choices: a block-level residual decomposition (§3.1), a frozen-backbone forward pass with a single learned correction injected only at the output (§3.2), and low-rank linear attention over depth via feature-factorized depth keys, fixed-size online state accumulation, and a single output readout (§3.3).

3.1 Block-Level Residual Decomposition

In post-training practice, depth in a decoder-only LLM is most commonly indexed at the block level: pretrained parameter layouts, PEFT placement, and default activation recomputation segments are organized around decoder blocks. LLARA therefore adopts the entire decoder block as the basic residual unit, in contrast with the layer-level view used by AttnRes in Section 2. Let $\boldsymbol{h}_0 \in \mathbb{R}^d$ be the input representation to the decoder stack and $\boldsymbol{h}_l \in \mathbb{R}^d$ the hidden state after block l, for $l = 1, \ldots, L$. A standard PreNorm decoder block (Xiong et al., 2020) stacks multi-head attention (MHA) and feed-forward network (FFN) (Vaswani et al., 2017), each applied after layer normalization (Ba et al., 2016) or RMSNorm (Zhang & Sennrich, 2019) and added back through residual connections (He et al., 2016). Collecting one entire block as a single residual increment, the recurrence reads:

$$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + \underbrace{\mathrm{MHA}_l(\mathrm{PN}_l^{(\mathrm{attn})}(\boldsymbol{h}_{l-1})) + \mathrm{FFN}_l\left(\mathrm{PN}_l^{(\mathrm{ffn})}\left(\boldsymbol{h}_{l-1} + \mathrm{MHA}_l(\mathrm{PN}_l^{(\mathrm{attn})}(\boldsymbol{h}_{l-1}))\right)\right)}_{\mathcal{F}_l(\boldsymbol{h}_{l-1})}, \tag{6}$$

where $\mathrm{MHA}_l, \mathrm{FFN}_l$ are the attention and feed-forward maps of block l, $\mathrm{PN}_l^{(\mathrm{attn})}$ and $\mathrm{PN}_l^{(\mathrm{ffn})}$ are the corresponding PreNorm normalizations, and $\mathcal{F}_l$ is the net residual contribution of block l.

Block-level homogeneity over depth. AttnRes treats each MHA and FFN sublayer as an independent residual unit, so the maps $\{f_l\}$ in Eq. (4) alternate between two distinct types and introduce a priori structural heterogeneity along depth. By contrast, LLARA bundles both sublayers into a single block increment $\mathcal{F}_l$, yielding structurally homogeneous updates $\{\mathcal{F}_l\}_{l=1}^L$ across depth.

3.2 Post-Training Attention Residual Adaptation

A direct port of AttnRes into a pretrained backbone would replace every intermediate hidden state by an attention-weighted aggregate of preceding residual values, following Eq. (4) inside the recurrence. Such a substitution is incompatible with post-training: the pretrained parameters were jointly optimized under Eq. (3), and inserting a new aggregator between blocks shifts the input distribution of every downstream block away from the regime its weights expect. This distributional mismatch can destabilize post-training optimization and induce immediate functional collapse, motivating a backbone-preserving interface. LLARA preserves the block-level residual recurrence in Eq. (6) throughout the forward pass, freezes the backbone weights, and augments the model with a single learned depth-wise correction applied only after the final block.

Concretely, let $\{\boldsymbol{h}_l\}_{l=0}^L$ be the trajectory produced by the frozen backbone, and define the realized block-level residual values:

$$\boldsymbol{v}_0 = \boldsymbol{h}_0, \quad \boldsymbol{v}_l = \boldsymbol{h}_l - \boldsymbol{h}_{l-1} = \mathcal{F}_l(\boldsymbol{h}_{l-1}), \quad l = 1, \ldots, L. \tag{7}$$

Rather than rewriting the recurrence, LLARA learns to map the realized increments $\{\boldsymbol{v}_l\}_{l=0}^L$ into a single correction vector $\Delta\boldsymbol{h} \in \mathbb{R}^d$ and produces the adapted final representation:

$$\tilde{\boldsymbol{h}}_L = \boldsymbol{h}_L + \Delta\boldsymbol{h}. \tag{8}$$

Under this design, intermediate blocks continue to emit exactly the sequence of inputs and outputs they were pretrained on, while the residual architecture is fine-tuned through how the realized increments are recomposed into the final representation that the language-modeling head consumes. The theoretical analysis motivating the additive readout in Eq. (8) is provided in §3.5.

3.3 Low-Rank Linear Attention over Depth

AttnRes evaluates a normalized exponential kernel between each layer and the preceding residual trajectory. When instantiated inside the recurrence as in Eq. (4), this requires a full depth-wise activation cache and depth-dependent pairwise scoring, making the mechanism poorly aligned with activation recomputation and imposing substantial GPU-memory overhead during post-training. LLARA instead performs feature-factorized linear attention[1] (Katharopoulos et al., 2020) over depth with low-rank depth keys, maintaining a compact linear-attention state with fixed-size online accumulators during the backbone forward pass and performing a single readout at the output.

Fixing a depth-attention rank $r \ll d$, LLARA assigns each depth position a learnable projection $\boldsymbol{A}_l \in \mathbb{R}^{r\times d}$, initialized at zero, and adopts the positive feature map:

$$\varphi(\boldsymbol{x}) = \mathrm{ELU}(\boldsymbol{x}) + 1. \tag{9}$$

At depth position l, the residual value $\boldsymbol{v}_l$ is mapped to a low-rank, strictly positive depth key:

$$\boldsymbol{k}_l = \varphi\left(\boldsymbol{A}_l\,\mathrm{RMSNorm}(\boldsymbol{v}_l)\right) \in \mathbb{R}^r_{>0}. \tag{10}$$

Following AttnRes, RMSNorm prevents residual scale from dominating depth weights. The projection $\boldsymbol{A}_l$ keeps values in $\mathbb{R}^d$ while moving the kernel to an r-dimensional positive feature space. This yields two fixed-size online accumulators for depth-wise mixing:

$$\boldsymbol{S}_l = \sum_{i=0}^{l} \boldsymbol{k}_i \boldsymbol{v}_i^\top \in \mathbb{R}^{r\times d}, \quad \boldsymbol{z}_l = \sum_{i=0}^{l} \boldsymbol{k}_i \in \mathbb{R}^r, \tag{11}$$

Starting from zero, they update with each emitted $\boldsymbol{v}_l$:

$$\boldsymbol{S}_l = \boldsymbol{S}_{l-1} + \boldsymbol{k}_l \boldsymbol{v}_l^\top, \quad \boldsymbol{z}_l = \boldsymbol{z}_{l-1} + \boldsymbol{k}_l. \tag{12}$$

Thus each token's depth history is compressed to $rd + r$ state variables, with no explicit residual list. After the final block, a learnable LLARA query $\boldsymbol{q} \in \mathbb{R}^r$, also initialized at zero, reads out the correction from $(\boldsymbol{S}_L, \boldsymbol{z}_L)$ using the same feature map:

$$\Delta\boldsymbol{h} = \frac{\boldsymbol{S}_L^\top \varphi(\boldsymbol{q})}{\varphi(\boldsymbol{q})^\top \boldsymbol{z}_L + \varepsilon} = \frac{\sum_{l=0}^{L} w_l \boldsymbol{v}_l}{\sum_{l=0}^{L} w_l + \varepsilon}, \quad w_l = \varphi(\boldsymbol{q})^\top \boldsymbol{k}_l, \tag{13}$$

where $\varepsilon > 0$ is a small constant for numerical stability. Equivalently, $\Delta\boldsymbol{h}$ is a normalized linear-attention aggregation of $\{\boldsymbol{v}_l\}_{l=0}^L$ with non-negative depth weights $\{w_l\}_{l=0}^L$ induced by the factorized kernel between the LLARA query $\boldsymbol{q}$ and the realized keys. The adapted hidden state $\tilde{\boldsymbol{h}}_L$ from Eq. (8) then replaces $\boldsymbol{h}_L$ at the language-modeling head:

$$p(y \mid x) = \mathrm{Softmax}\left(\boldsymbol{W}_{\mathrm{lm}}\mathrm{Norm}_{\mathrm{out}}(\tilde{\boldsymbol{h}}_L)\right). \tag{14}$$

This design preserves the pretrained block interface exactly, since no intermediate block receives a modified input. Although the LLARA query $\boldsymbol{q}$ is a global readout parameter, the depth weights $\{w_l\}_{l=0}^L$ remain input-dependent because the keys $\{\boldsymbol{k}_l\}_{l=0}^L$ are computed from the token's realized hidden trajectory through the frozen backbone. LLARA thereby achieves input-dependent selective composition over depth without rewriting the backbone recurrence.

3.4 Parameter-Efficient Fine-Tuning and Complexity

Joint Training with LoRA. LLARA is trained jointly with LoRA (Hu et al., 2022) adapters inserted into selected linear submodules of the frozen backbone. Denoting the collection of LoRA low-rank matrices by Θ_{LoRA}, the trainable parameter set is $\Theta_{\mathrm{train}} = \Theta_{\mathrm{LoRA}} \cup \{\boldsymbol{A}_l\}_{l=0}^L \cup \{\boldsymbol{q}\}$. LoRA and LLARA act on complementary axes of the architecture: LoRA modifies the within-block transformation $\mathcal{F}_l$ of each decoder block, whereas LLARA modifies the across-block composition $\{\boldsymbol{v}_l\}_{l=0}^L \mapsto \Delta\boldsymbol{h}$ of the realized residual increments. Since the two mechanisms share no parameters and operate at different structural scales, they can be optimized jointly without interference, enabling simultaneous parameter fine-tuning and architectural fine-tuning of the pretrained LLMs. Appendix H discusses why LLARA is trained jointly with LoRA rather than as a standalone adapter.

[1] See Appendix D for details of linear attention.

Complexity Analysis. The LLARA-specific state per token consists only of $\boldsymbol{S}_l \in \mathbb{R}^{r\times d}$ and $\boldsymbol{z}_l \in \mathbb{R}^r$, contributing $O(rd)$ additional memory that is independent of the number of decoder blocks. Maintaining this state across all L blocks costs $O(Lrd)$ additional arithmetic, plus a single $O(rd)$ readout at the output. By contrast, a full softmax attention-residual cache must retain every depth value and evaluate $O(L)$ pairwise scores at every depth, incurring depth-dependent storage and pairwise attention work. Choosing $r \ll d$ therefore yields a compact, post-training-compatible residual adaptation that preserves the frozen backbone's forward interface while still providing expressive, input-dependent control over how realized residual increments are composed across depth. Appendix J.2 reports training time, GPU memory, and related computational cost.

3.5 Theoretical Analysis in A Gradient Perspective

Under SGD (Robbins & Monro, 1951), we compare the gradient structure of LLARA and AttnRes, showing that LLARA admits more stable architectural gradients under a frozen backbone. For a controlled comparison, we analyze both schemes at the block-residual level in Eq. (7): LLARA reads out the frozen realized increments, whereas AttnRes inserts a depth-attention aggregate into the recurrence and is bounded on its own realized trajectory. Without loss of generality, we analyze SGD rather than AdamW (Loshchilov & Hutter, 2019), since AdamW maps loss gradients to parameter updates via adaptive reweighting, momentum, and decoupled weight decay, and therefore does not alter the structural form of those gradients established in Theorem 1 and Appendix G. Fixing a token, a scalar loss $\mathcal{L}$ routed through Eq. (14), and the block-level AttnRes and LLARA readout maps (Eqs. (74) and (76)), we obtain the following comparison.

Theorem 1 (Architectural gradient comparison)**.** *Under Assumptions 14 and 15 in Appendix G, with both mechanisms instantiated at the same block-residual granularity over frozen block maps:*

(i) For AttnRes, let $\boldsymbol{M}_l = \partial \boldsymbol{h}_l / \partial \boldsymbol{q}_l$ be the local aggregate-to-query Jacobian, holding $\{\boldsymbol{v}_i\}_{i=0}^{l}$ fixed when differentiating the softmax map at block l, and let $\bar{\boldsymbol{g}}_l$ be the $\boldsymbol{h}_l$ component of the total state adjoint at depth l. Then $\nabla_{\boldsymbol{q}_l}\mathcal{L} = \boldsymbol{M}_l^\top \bar{\boldsymbol{g}}_l$, and

$$\|\nabla_{\boldsymbol{q}_l}\mathcal{L}\|_2 \le \Big(\prod_{m=l+1}^{L} L_{\mathrm{AttnRes}}\Big)\, C_{\mathrm{loc}}\, G, \quad C_{\mathrm{loc}} := 2\sqrt{d}\, V. \tag{15}$$

At $\boldsymbol{q}_l = \boldsymbol{0}$, the weights satisfy $\alpha_{i\to l} = 1/(l{+}1)$, and $\boldsymbol{M}_l^\top \bar{\boldsymbol{g}}_l$ can be non-zero whenever $\boldsymbol{M}_l \neq \boldsymbol{0}$ and $\bar{\boldsymbol{g}}_l \neq \boldsymbol{0}$.

(ii) For LLARA, architecture gradients pass only through the single readout in Eq. (13). The key map is depth-local, since $\partial \boldsymbol{k}_i / \partial \boldsymbol{A}_l = \boldsymbol{0}$ for $i \neq l$, while the normalized denominator still couples all depths. For the LLARA query, and for every $l = 0, \dots, L$,

$$\|\nabla_{\boldsymbol{A}_l}\mathcal{L}\|_F \le \frac{2B_\varphi\, \kappa G V \sqrt{d}}{d_{\min}}, \quad \|\nabla_{\boldsymbol{q}}\mathcal{L}\|_2 \le \frac{2B_\varphi\,(L+1)\kappa G V}{d_{\min}}, \tag{16}$$

with no recurrent Jacobian product along depth. The factor $L+1$ in the LLARA-query bound comes only from the single output-level readout over all depths. At $\boldsymbol{A}_l = \boldsymbol{0}$ and $\boldsymbol{q} = \boldsymbol{0}$, one has $\boldsymbol{k}_l = \boldsymbol{u} = \boldsymbol{1}$ and $w_l = r$, so the readout is a depth-uniform aggregate before learning; with $\varepsilon = 0$, $\nabla_{\boldsymbol{q}}\mathcal{L} = \boldsymbol{0}$ at this point because $\sum_{i=0}^{L}(\boldsymbol{v}_i - \Delta\boldsymbol{h}) = \boldsymbol{0}$.

Theorem 1 isolates why LLARA admits more stable architectural gradients when adapting a frozen backbone. AttnRes places its trainable queries inside the depth recurrence, so the query gradient at block l is modulated by the total adjoint from later blocks and can carry products of effective state-transition sensitivities up to depth L. LLARA is more stable because its architecture parameters are outside this recurrence. Once the frozen residual trajectory is realized, gradients to $\{\boldsymbol{A}_l\}$ and $\boldsymbol{q}$ pass through only the final normalized readout, so their path length does not grow with the number of later blocks. The per-depth projections $\boldsymbol{A}_l$ admit depth-local bounds controlled by $G, V, \kappa, d_{\min}$, and B_φ, rather than by products of recurrent Jacobians along the remaining $L - l$ blocks. The LLARA query $\boldsymbol{q}$ still couples all depths through the normalized readout and therefore carries an output-level $(L+1)$ summation factor, but this is not a repeated product through backbone transitions. Proofs and gradient derivations are given in Appendix G.

4 EXPERIMENTS

In this section, we evaluate whether LLARA improves downstream adaptation of pretrained LLMs under a parameter-efficient fine-tuning paradigm, addressing the following six research questions:

Q1: Does LLARA improve domain-specific supervised fine-tuning over a matched LoRA baseline across knowledge, mathematical reasoning, and code generation? (Section 4.2)
Q2: Does LLARA remain beneficial under sparse verifiable-reward reinforcement learning with different policy-optimization algorithms such as PPO and GRPO? (Section 4.2)
Q3: Does LLARA remain effective across different models and parameter scales? (Section 4.3)
Q4: Do LLARA's training dynamics and learned depth readouts empirically support the architectural gradient-stability and selective residual-aggregation claims of our analysis? (Section 4.4)
Q5: Is LLARA sensitive to the depth-attention rank hyperparameter r? (Appendix J.1)
Q6: Is LLARA parameter-efficient relative to the matched LoRA baseline? (Appendix J.2)

4.1 EXPERIMENTAL SETUP

SFT tasks. We evaluate LLARA via domain-specific SFT of base Qwen2.5-7B (Qwen Team, 2024), training one adapter per domain on instruction-formatted corpora. General-domain SFT uses 100K examples from the Tülu 3 SFT mixture (Lambert et al., 2024) and is evaluated on MMLU (Hendrycks et al., 2021a), MMLU-Pro (Wang et al., 2024), and TriviaQA (Joshi et al., 2017); mathematical SFT uses 100K MetaMathQA examples (Yu et al., 2024) and is evaluated on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b); and code SFT uses Magicoder-Evol-Instruct-110K (Wei et al., 2024), evaluated on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). In the mathematical instruction tuning setting, we additionally evaluate LLARA on Gemma-7B (Gemma Team, 2024) and Llama-3.1-8B (Llama Team, 2024). We report multiple-choice accuracy on MMLU and MMLU-Pro, exact-match answer accuracy on TriviaQA, exact-match-style answer accuracy on GSM8K and MATH, and pass@1 on HumanEval and MBPP.

RL tasks. For reinforcement learning, we fine-tune base Qwen2.5-3B (Qwen Team, 2024) on the GSM8K training split (Cobbe et al., 2021) with verifiable rewards based on final-answer correctness, using Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Group Relative Policy Optimization (GRPO) (Shao et al., 2024) as two distinct post-training strategies. We evaluate LLARA under both algorithms on GSM8K and MATH (Hendrycks et al., 2021b).

Baseline. We adopt LoRA (Hu et al., 2022) as the baseline because LLARA is an architecture-adaptation paradigm built on parameter-efficient fine-tuning rather than a replacement for it. Specifically, we compare a matched LoRA-only baseline to joint LoRA and LLARA training under identical adapter placement, LoRA hyperparameters, and optimization budget, isolating the benefit of adapting cross-block residual aggregation. Further details on configuration, dataset preprocessing, evaluation protocols, and hyperparameters are provided in Appendix I, and the PEFT block-wrapper implementation of LLARA is described in Appendix F.

4.2 MAIN RESULTS

Supervised fine-tuning. As shown in Table 1, LLARA improves over the matched baseline on every domain-specific SFT benchmark spanning knowledge, mathematical reasoning, and code generation. The advantage is not confined to a single task family: knowledge-oriented evaluations such as MMLU-Pro (+1.4) and TriviaQA (+1.3) improve alongside mathematical reasoning such as GSM8K (+2.3) and MATH (+2.5), and code generation such as MBPP (+2.8). This breadth suggests that learning token-dependent cross-block residual composition provides a complementary pathway for instruction fine-tuning beyond within-block parameter adaptation.

Reinforcement learning. We next evaluate whether depth-wise residual adaptation remains useful under reward-driven post-training, where sparse sequence-level feedback replaces supervised next-token likelihood. On Qwen2.5-3B, LLARA improves the matched baseline under PPO on GSM8K (+1.6) and MATH (+2.4), and under GRPO on GSM8K (+2.4) and MATH (+2.9). These gains under both PPO and GRPO suggest that cross-block residual composition remains effective when post-training shifts from supervised likelihood to sparse sequence-level rewards.

Table 1: Performance comparison of LLARA with the baseline under SFT and RL. Best per-row results are **bolded**.

Domain	Task	Model	Benchmark	Accuracy (%)	
				Baseline	LLARA
General	SFT	Qwen2.5-7B	MMLU	71.6	**72.5**
			MMLU-Pro	44.3	**45.7**
			TriviaQA	56.9	**58.2**
Math	SFT	Qwen2.5-7B	GSM8K	82.0	**84.3**
			MATH	47.4	**49.9**
	SFT	Gemma-7B	GSM8K	72.9	**75.3**
			MATH	27.6	**29.0**
	SFT	Llama-3.1-8B	GSM8K	73.7	**74.7**
			MATH	25.9	**26.8**
	RL(PPO)	Qwen2.5-3B	GSM8K	73.8	**75.4**
			MATH	37.2	**39.6**
	RL(GRPO)	Qwen2.5-3B	GSM8K	74.1	**76.5**
			MATH	37.9	**40.8**
Code	SFT	Qwen2.5-7B	HumanEval	57.5	**59.1**
			MBPP	62.4	**65.2**

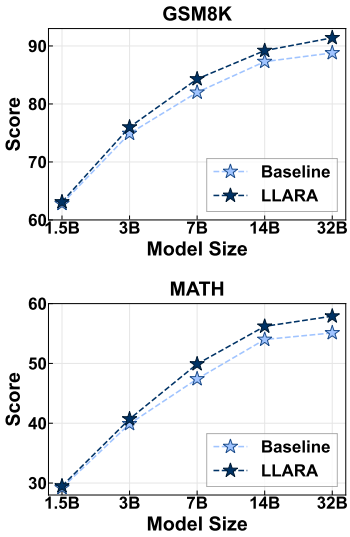


Figure 2: Scaling of LLARA on Qwen2.5 from 1.5B to 32B.

4.3 LLARA across Backbones and Scales

Different backbones. Table 1 further reports mathematical SFT on Gemma-7B and Llama-3.1-8B alongside Qwen2.5-7B. Despite distinct pretraining recipes, all three adopt residual connections with PreNorm. Under matched PEFT placement, LLARA rank, and training budget, LLARA improves both GSM8K and MATH on every backbone. This consistency indicates that LLARA transfers across PreNorm residual LLMs, conferring token-dependent cross-block residual composition during post-training rather than remaining tied to a single pretrained residual geometry.

Different parameter scales. Figure 2 evaluates LLARA on Qwen2.5 from 1.5B to 32B under the same math SFT protocol. The improvement over the matched baseline is modest at the smallest scale and becomes more pronounced as model size increases on both GSM8K and MATH. If LLARA were merely compensating for limited backbone capacity, the gap should shrink as the pretrained model grows. The opposite trend suggests that uniform residual aggregation leaves a depth-mixing degree of freedom that within-block adaptation (e.g., LoRA) does not occupy, and that this leftover becomes more consequential at larger scale. LLARA occupies this axis by learning token-dependent cross-block residual composition during post-training.

4.4 Training Dynamics and Learned Depth Selectivity

Why in-place AttnRes fails. We exclude AttnRes from the preceding comparisons because it is infeasible during post-training. The matched in-place AttnRes retains the same within-block LoRA adapters while replacing the pretrained residual connections with block-level AttnRes (Eq. 74) during fine-tuning. Empirically, even with gradient clipping, both the training loss and the gradient norm of AttnRes remain several orders of magnitude larger than those of LLARA, rendering effective fine-tuning infeasible (Figure 3). This failure mode reflects a structural mismatch with the pretrained backbone, as inserting AttnRes between blocks immediately shifts every downstream block input from the uniform residual-aggregation regime under which those weights were optimized. This is consistent with Theorem 1(i): AttnRes query updates propagate through depth-recurrent Jacobian products. Within-block LoRA can retune maps $\mathcal{F}_l$, but cannot restore a well-conditioned depth-attention geometry once the residual operator is rewritten inside the depth recurrence.

Training dynamics of LLARA. Figure 3 also compares the matched LoRA baseline and LLARA on Qwen2.5-7B mathematical SFT, with training loss and global gradient norm shown after exponential-moving-average smoothing. The two loss curves remain closely aligned throughout training, including the drop at the start of the second epoch, and the minimum logged loss is slightly

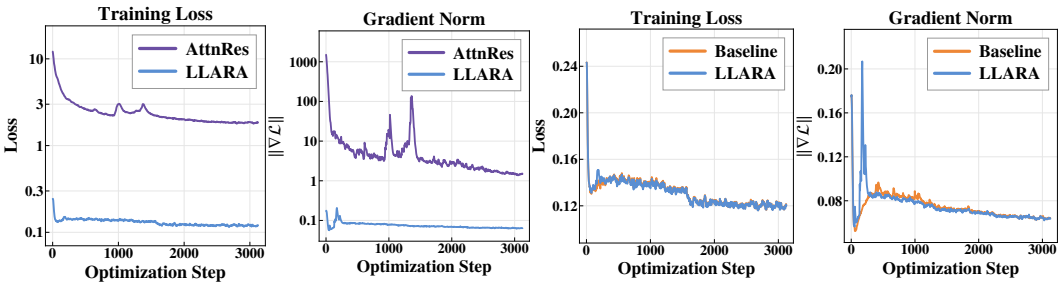


Figure 3: Training dynamics on Qwen2.5-7B mathematical SFT.

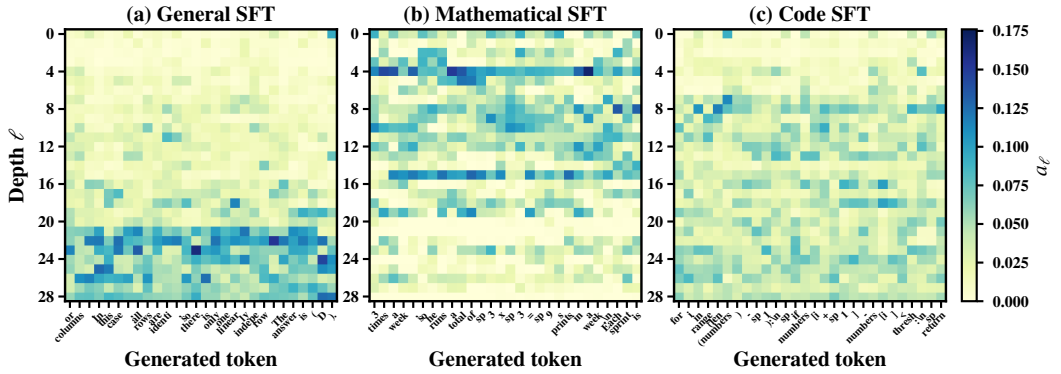


Figure 4: Token-wise depth aggregation after post-training Qwen2.5-7B. **(a)** General SFT on an MMLU-Pro test completion. **(b)** Mathematical SFT on a GSM8K test completion. **(c)** Code SFT on a HumanEval test completion. Each column is the normalized LLARA weight a_l.

lower for LLARA (0.0916) than for the baseline (0.1078). The gradient norm of LLARA is more variable in the first few hundred steps, then gradually settles and tracks the baseline rather than diverging. That the trajectory restabilizes is consistent with the backbone-preserving design: LLARA leaves the pretrained residual recurrence intact and applies a trainable depth readout only after the final block, so architectural gradients follow a single-readout path instead of AttnRes-style recurrent Jacobian products (Theorem 1). Overall, LLARA can be trained stably under the same post-training protocol as the baseline, in line with the gradient analysis in §3.5.

Selective residual aggregation. To inspect the mechanism learned by LLARA, we recover the normalized contribution of depth l for each token as $a_l = w_l/(\sum_{j=0}^{L} w_j + \varepsilon)$, where $w_l = \varphi(\boldsymbol{q})^\top \boldsymbol{k}_l$. Figure 4 visualizes these coefficients over generated tokens and decoder depth. Although $\boldsymbol{q}$ is shared, each $\boldsymbol{k}_l$ depends on the token-specific residual increment $\boldsymbol{v}_l$. Consequently, the heatmap tests the input-dependent selectivity claimed by Eq. (13). On Qwen2.5-7B, the three domains induce different depth profiles. General-domain completions concentrate mass on late decoder blocks, mathematical completions emphasize a small set of early-to-middle blocks, and code completions spread weight more evenly into later layers and vary more strongly with token role. Appendix J.3 reports full-completion traces across domains and compares the mathematical depth profile across backbones.

5 Conclusion

In this paper, we proposed LLARA, a parameter-efficient post-training recipe that introduces token-wise attention residuals into pretrained LLMs without rewriting their residual recurrence. Uniform residual accumulation under PreNorm induces depth-wise dilution, yet Attention Residuals that alleviate this issue have required pre-training from scratch. LLARA freezes the backbone and, jointly with LoRA, learns input-dependent cross-block residual composition through a compact low-rank linear-attention pathway. Experiments show consistent gains over a matched LoRA baseline across knowledge, mathematical reasoning, and code generation under both supervised and reinforcement post-training, with the advantage transferring across PreNorm residual backbones.

AI USE STATEMENT

In the preparation of this work, we use large language models (LLMs) to assist with proofreading, grammatical correction, and language polishing. The LLM serves solely as a tool to enhance the clarity and readability of our writing. We meticulously review and edit all AI-generated content, and we accept full responsibility for the final version of the manuscript.

REPRODUCIBILITY STATEMENT

We provide implementation details, training protocols, and evaluation settings to support reproduction. The LLARA parameterization and PEFT block-wrapper interface are specified in §3 and Appendix F. Theorem 1 is proved in Appendix G. Hardware, software, datasets, hyperparameters, and evaluation protocols are reported in Appendix I, with additional results and ablations in Appendix J.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. ReZero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, William Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations*, 2016.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Rei Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Róbert Csordás, Christopher D. Manning, and Christopher Potts. Do language models use their depth efficiently? In *Advances in Neural Information Processing Systems*, 2025.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Gemma Team. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 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*, 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*, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021b.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*. PMLR, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2017.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*. PMLR, 2020.

Kimi Team. Attention Residuals. *arXiv preprint arXiv:2603.15031*, 2026a.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026b.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Tim Lawson, Lucy Farnik, Conor Houghton, and Laurence Aitchison. Residual stream analysis with multi-layer SAEs. In *International Conference on Learning Representations*, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, 2022.

Tianyu Li, Dongchen Han, Zixuan Cao, Haofeng Huang, Mengyu Zhou, Ming Chen, Erchao Zhao, Xiaoxi Jiang, Guanjun Jiang, and Gao Huang. SiameseNorm: Breaking the barrier to reconciling Pre/Post-norm. *arXiv preprint arXiv:2602.08064*, 2026.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, 2021.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, 2022.

Llama Team. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

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*, Vienna, Austria, 2025. Association for Computational Linguistics.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In *Advances in Neural Information Processing Systems*, 2024.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. LST: Ladder side-tuning for parameter and memory efficient transfer learning. In *Advances in Neural Information Processing Systems*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. DeepNet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems Track on Datasets and Benchmarks*, 2024.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In *International Conference on Machine Learning*. PMLR, 2024.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. ReFT: Representation finetuning for language models. In *Advances in Neural Information Processing Systems*, 2024.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mHC: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Huishuai Xing, Yuxiang Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*. PMLR, 2020.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, 2024.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.

Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Yu Qiao, Hongsheng Li, and Peng Gao. LLaMA-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *International Conference on Learning Representations*, 2024.

Yifan Zhang, Yifeng Liu, Mengdi Wang, and Quanquan Gu. Deep delta learning. *arXiv preprint arXiv:2601.00417*, 2026.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *International Conference on Learning Representations*, 2025.

APPENDIX

A READING GUIDE

Reading Guideline. The appendix is organized as follows. **Appendix B** reviews depth-wise residual aggregation and post-training fine-tuning paradigms, and locates LLARA relative to both. **Appendix C** isolates structural conditions under which uniform residual accumulation dilutes task-relevant increments, and shows that normalized depth-wise attention can preserve them. **Appendix D** reviews feature-factorized linear attention, the computational primitive behind the LLARA readout. **Appendix E** recalls the LoRA parameterization and explains why LoRA and LLARA act on complementary parts of the computation graph. **Appendix F** maps the method to the PEFT block-wrapper implementation, including the forward state, training interface, and checkpointing compatibility. **Appendix G** derives the block-level gradients of attention residuals and of LLARA, and proves Theorem 1. **Appendix H** explains why LLARA is trained jointly with LoRA rather than as a standalone adapter. **Appendix I** reports hardware, software, datasets, hyperparameters, and evaluation protocols. **Appendix J** provides extended benchmark results, rank-sensitivity ablations, efficiency measurements, and additional depth-weight visualizations.

Notation. A summary of the primary notation used is provided in the table below. It lists the recurring objects of the model, the LLARA mechanism, and the theoretical analysis. Purely local constants and standard mathematical symbols are defined where they first appear.

Table 2: Primary Notation.

Notation	Description
$\boldsymbol{h}_l$	Hidden state after layer or block l; $\boldsymbol{h}_0$ the token embedding
$\tilde{\boldsymbol{h}}_L, \Delta\boldsymbol{h}$	Adapted final state $\boldsymbol{h}_L + \Delta\boldsymbol{h}$ and depth-readout correction
f_l	Sublayer map (MHA or FFN) in the layer-level view
$\mathcal{F}_l$	Net residual map of decoder block l
$\boldsymbol{v}_l$	Residual value; $\boldsymbol{v}_0 = \boldsymbol{h}_0$ and $\boldsymbol{v}_l = \boldsymbol{h}_l - \boldsymbol{h}_{l-1}$
$\mathcal{A}_l(a)$	Depth-wise aggregation $\sum_{i=0}^{l} a_i \boldsymbol{v}_i$
$\alpha_{i\to l}$	AttnRes softmax depth weight from i to l
$\boldsymbol{q}_l$	AttnRes layer-specific pseudo-query, zero-initialized
$\tilde{\boldsymbol{k}}_i$	Normalized AttnRes key $\mathrm{RMSNorm}(\boldsymbol{v}_i)$
φ	Positive feature map $\varphi(x) = \mathrm{ELU}(x) + 1$
$\boldsymbol{A}_l, \boldsymbol{k}_l$	Depth projection (zero-init) and key $\varphi(\boldsymbol{A}_l \mathrm{RMSNorm}(\boldsymbol{v}_l))$
$\boldsymbol{q}, \boldsymbol{u}$	LLARA readout query (zero-init) and feature $\boldsymbol{u} = \varphi(\boldsymbol{q})$
$\boldsymbol{S}_l, \boldsymbol{z}_l$	Online accumulators $\sum_{i\le l} \boldsymbol{k}_i \boldsymbol{v}_i^\top$ and $\sum_{i\le l} \boldsymbol{k}_i$
w_l	Unnormalized depth weight $\boldsymbol{u}^\top \boldsymbol{k}_l$
$\mathrm{Norm}_{\mathrm{out}}, \boldsymbol{W}_{\mathrm{lm}}$	Output normalization and language-modeling head
$\Gamma_i^{\mathrm{std}}(l), \Gamma_{\boldsymbol{p}}(\boldsymbol{y})$	Relative and directional visibility along probe $\boldsymbol{p}$
S, ρ	Informative depths (signal μ) and attention mass on S
$\boldsymbol{\xi}_i$	Zero-mean noise in $\boldsymbol{v}_i$, with scale bound τ
$\Lambda_\lambda(a)$	Decay-weighted pairwise noise $\sum_{i<j} a_i a_j \lambda^{\lvert i-j\rvert}$
$\boldsymbol{\zeta}_m, \mathcal{T}_m, \boldsymbol{J}_m$	AttnRes state, transition, and Jacobian
$\boldsymbol{M}_l$	Local Jacobian $\partial \boldsymbol{h}_l / \partial \boldsymbol{q}_l$
$\bar{\boldsymbol{\eta}}_l, \bar{\boldsymbol{g}}_l$	AttnRes state adjoint and its $\boldsymbol{h}_l$ component
$\boldsymbol{B}_m, \boldsymbol{R}_i$	Block Jacobian of $\mathcal{F}_m$ and RMSNorm Jacobian
$\boldsymbol{g}$	Upstream gradient $\nabla_{\Delta\boldsymbol{h}} \mathcal{L}$

B RELATED WORK

Depth-Wise Residual Aggregation. Depth-wise residual aggregation determines how residual increments are composed along decoder depth, with the standard recurrence in Eq. (1) corresponding to a unit-weight sum. Highway Networks (Srivastava et al., 2015) gate the identity shortcut, ReZero (Bachlechner et al., 2020) introduces zero-initialized residual scalars, and DeepNet (Wang et al., 2022) uses depth-dependent scaling and initialization to stabilize deep Transformers. PreNorm (Xiong et al., 2020) improves gradient flow by normalizing before each residual

branch, while SiameseNorm (Li et al., 2026) reconciles PreNorm and PostNorm through a paired normalization design. Beyond stabilization, DenseFormer (Pagliardini et al., 2024) feeds blocks with depth-weighted averages, Hyper-Connections (Zhu et al., 2025) and mHC (Xie et al., 2025) replace scalar residual coefficients with structured learnable connections, and Deep Delta Learning (Zhang et al., 2026) emphasizes incremental delta signals. Attention Residuals (Kimi Team, 2026a) further replace fixed accumulation with token-wise softmax attention over depth, but the resulting layer interface is learned during pretraining and is not directly compatible with post-training existing LLMs. These designs improve how residual information is aggregated when the architecture is trained, but leave open how such depth-wise control can be introduced into an already pretrained backbone during post-training.

Fine-Tuning Paradigms. Post-training adaptation of pretrained LLMs can be organized by where trainable task-specific capacity is attached to the frozen model. LoRA (Hu et al., 2022) parameterizes low-rank updates to existing linear maps and can be merged into the base weights at inference, without introducing a new architectural pathway. Module- and activation-level methods operate by inserting or modulating computation inside Transformer blocks: Adapters (Houlsby et al., 2019) insert bottleneck residual modules inside Transformer blocks, $(IA)^3$ (Liu et al., 2022) rescales attention and feed-forward activations with learned vectors, and LLaMA-Adapter (Zhang et al., 2024) injects learnable adaptation prompts through zero-initialized attention gates. Prefix-Tuning (Li & Liang, 2021) modifies the attention interface by prepending trainable key-value prefixes at each layer. Ladder Side-Tuning (Sung et al., 2022) attaches a lightweight side network to intermediate backbone activations, while ReFT (Wu et al., 2024) learns localized interventions on hidden representations. These paradigms adapt weights, block-internal transformations, attention inputs, side paths, or representation slices, but none targets the cross-block residual aggregation rule itself.

LLARA introduces a new paradigm for model architecture fine-tuning by adapting the residual aggregation structure of a pretrained LLM. Rather than modifying block-internal transformations, attention inputs, representation slices, or side networks, LLARA fine-tunes how block-level residual contributions are recomposed at the output. It preserves the pretrained block interface, maintains a fixed-size low-rank linear-attention state over the residual trajectory, and learns a token-dependent correction to the final representation. In this sense, LLARA transfers the selective depth-mixing principle of attention residuals to post-training and turns cross-block residual aggregation into a trainable architectural component of existing LLMs.

C Why Attention Residuals Improve Depth-Wise Aggregation

This section provides a mathematical justification for replacing uniform unweighted residual accumulation with normalized depth-wise attention. The purpose of the analysis is not to claim that attention residuals universally outperform standard residuals in all regimes. Rather, we isolate structural conditions motivated by deep PreNorm residual streams: residual increments have depth-stable scale, different increments are weakly aligned, and task-relevant information may be concentrated in a small subset of depth positions for a given token or task. Under these conditions, standard residuals exhibit depth-wise dilution, whereas normalized attention residuals can preserve task-relevant increments and downweight uninformative depth positions. We also make explicit why heterogeneous layer maps f_l can make depth-wise attention harder to learn, motivating the block-level residual values used by LLARA.

We begin by casting both standard residual accumulation and normalized depth-wise attention as instances of a common aggregation operator.

Definition 2 (Depth-wise aggregation)**.** Fix a depth l and residual values $\{\boldsymbol{v}_i\}_{i=0}^{l} \subset \mathbb{R}^d$. For a weight vector $a = (a_0, \ldots, a_l)$, we define the depth-wise aggregation as:

$$\mathcal{A}_l(a) = \sum_{i=0}^{l} a_i \boldsymbol{v}_i. \tag{17}$$

The standard residual stream corresponds to the unnormalized unit weights $a_i = 1$, since $\boldsymbol{h}_l = \sum_{i=0}^{l} \boldsymbol{v}_i$. A normalized depth-attention residual corresponds to $a_i \geq 0$ and $\sum_{i=0}^{l} a_i = 1$, with $a_i = \alpha_{i\to l}$ for softmax AttnRes in Eq. (4). For the LLARA readout in Eq. (13), the implementation includes $\varepsilon > 0$ only to avoid a numerically singular denominator. Throughout the theoretical analysis, we ignore this stabilizer and use the normalized weights $a_i = w_i / \sum_{j=0}^{l} w_j$.

RMSNorm convention. Throughout the theoretical analysis, $\mathrm{RMSNorm}(\boldsymbol{v})$ denotes the standard root-mean-square normalization used in AttnRes scoring and LLARA depth keys:

$$\mathrm{rms}(\boldsymbol{v}) = \sqrt{\frac{1}{d}\|\boldsymbol{v}\|_2^2 + \delta}, \quad \mathrm{RMSNorm}(\boldsymbol{v}) = \frac{\boldsymbol{v}}{\mathrm{rms}(\boldsymbol{v})}, \tag{18}$$

with stability constant $\delta > 0$. For every $\boldsymbol{v} \in \mathbb{R}^d$, one has $\|\mathrm{RMSNorm}(\boldsymbol{v})\|_2^2 = \|\boldsymbol{v}\|_2^2/(\|\boldsymbol{v}\|_2^2/d + \delta) \le d$, hence $\|\mathrm{RMSNorm}(\boldsymbol{v})\|_2 \le \sqrt{d}$. We write $\mathrm{Norm}_{\mathrm{out}}$ for the same map applied to the final hidden state before the language-modeling head in Eq. (14).

The subsequent analysis rests on assumptions motivated by empirical and architectural findings about PreNorm residual streams, which we justify below as reasonable.

Assumption 3 (Depth-stable weakly aligned increments)**.** Consistent with the PreNorm residual-stream analysis in Section 1, fix a token and let $\boldsymbol{v}_i = f_i(\boldsymbol{h}_{i-1})$ denote the residual increment at depth i. There exist constants $\sigma > 0$, $0 < c \le C < \infty$, and $0 \le \eta < c$, all independent of l, such that for all $1 \le i \le l$:

$$c\sigma^2 \le \mathbb{E}\|\boldsymbol{v}_i\|_2^2 \le C\sigma^2. \tag{19}$$

Moreover, the increments are weakly aligned in the sense that:

$$\left| 2 \sum_{1 \le i < j \le l} \mathbb{E}\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle \right| \le \eta l \sigma^2. \tag{20}$$

Why is Assumption 3 reasonable? Assumption 3 is a coarse second-moment condition for trained PreNorm residual streams rather than an independence or identical-distribution claim. The scale bound follows from the role of PreNorm in modern Transformers: LayerNorm/RMSNorm normalizes the input to each attention or feed-forward branch, so residual increments are not expected to vanish merely because the network is deeper (Xiong et al., 2020; Ba et al., 2016; Zhang & Sennrich, 2019). The weak-alignment bound is similarly reasonable. It does not require pairwise orthogonality of the increments. It only excludes the degenerate case where most depth increments point coherently in a common direction, which would make the accumulated cross terms grow quadratically in l. This exclusion is consistent with high-dimensional residual streams, where heterogeneous blocks compute different updates and empirical studies show that useful layer contributions are uneven and often token- or task-dependent rather than copies of a single direction (Gromov et al., 2025; Men et al., 2025; Csordás et al., 2025; Lawson et al., 2025). Thus, Assumption 3 describes the conservative random-walk regime for standard residual accumulation. If trained increments are more positively aligned, the residual norm grows faster, which strengthens rather than weakens the case for normalized depth-wise aggregation.

Lemma 4 (Random-walk growth of standard residuals)**.** *Under Assumption 3, the cumulative residual deviation satisfies:*

$$(c - \eta) l \sigma^2 \le \mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 \le (C + \eta) l \sigma^2. \tag{21}$$

In particular, $\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 = \Theta(l\sigma^2)$.

Proof. By unrolling the standard residual recurrence in Eq. (1) and using Eq. (2), we obtain:

$$\boldsymbol{h}_l - \boldsymbol{h}_0 = \sum_{i=1}^{l} \boldsymbol{v}_i. \tag{22}$$

Expanding the squared norm gives:

$$\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 = \sum_{i=1}^{l} \mathbb{E}\|\boldsymbol{v}_i\|_2^2 + 2 \sum_{1 \le i < j \le l} \mathbb{E}\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle. \tag{23}$$

The first term is bounded between $cl\sigma^2$ and $Cl\sigma^2$ by Eq. (19). The absolute value of the second term is bounded by $\eta l \sigma^2$ by Eq. (20). Combining the lower bounds yields:

$$\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 \ge cl\sigma^2 - \eta l \sigma^2 = (c - \eta) l \sigma^2, \tag{24}$$

and combining the upper bounds yields:

$$\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 \le Cl\sigma^2 + \eta l \sigma^2 = (C + \eta) l \sigma^2. \tag{25}$$

Since $0 \le \eta < c$, we have $c - \eta > 0$, so both bounds are proportional to $l\sigma^2$ with constants independent of l, and hence $\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2 = \Theta(l\sigma^2)$. □

To quantify the depth-wise dilution of individual increments relative to this growing aggregate, we next introduce relative visibility.

Definition 5 (Relative visibility)**.** We formalize the depth-wise dilution and relative contribution in Section 1 through amplitude-level visibility. Each quantity is a ratio of root-mean-square magnitudes computed from marginal second moments, rather than a second-moment share such as $\mathbb{E}\|\boldsymbol{v}_i\|_2^2/\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2$. For a residual increment $\boldsymbol{v}_i$ in the standard residual stream, its relative visibility at depth l is:

$$\Gamma_i^{\mathrm{std}}(l) = \frac{\left(\mathbb{E}\|\boldsymbol{v}_i\|_2^2\right)^{1/2}}{\left(\mathbb{E}\|\boldsymbol{h}_l - \boldsymbol{h}_0\|_2^2\right)^{1/2}}. \tag{26}$$

For a unit direction $\boldsymbol{p} \in \mathbb{R}^d$ and a depth-wise aggregation output $\boldsymbol{y}$, directional visibility is:

$$\Gamma_{\boldsymbol{p}}(\boldsymbol{y}) = \frac{|\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}\rangle|}{\left(\mathbb{E}\|\boldsymbol{y}\|_2^2\right)^{1/2}}. \tag{27}$$

Thus $\Gamma_i^{\mathrm{std}}(l)$ quantifies increment-level dilution relative to the cumulative deviation $\boldsymbol{h}_l - \boldsymbol{h}_0$, and $\Gamma_{\boldsymbol{p}}(\boldsymbol{y})$ quantifies the normalized projection of an aggregated output onto $\boldsymbol{p}$.

Theorem 6 (Depth-wise dilution of standard residuals)**.** *Under Assumption 3, every fixed residual increment satisfies the relative-visibility bound:*

$$\Gamma_i^{\mathrm{std}}(l) \le \sqrt{\frac{C}{c-\eta}}\frac{1}{\sqrt{l}}, \quad 1 \le i \le l. \tag{28}$$

This makes explicit the $O(1/\sqrt{l})$ relative-contribution scaling in Section 1. Moreover, fix a unit direction $\boldsymbol{p} \in \mathbb{R}^d$ and an informative set $S \subseteq \{1, \dots, l\}$ with $|S| = m$. Suppose the directional means satisfy:

$$|\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i\rangle| \le M \quad \textit{for } i \in S, \quad \mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i\rangle = 0 \quad \textit{for } i \notin S, \tag{29}$$

where m and M do not grow with l. Then the directional visibility of the cumulative residual deviation $\boldsymbol{y}_l^{\mathrm{std}} = \boldsymbol{h}_l - \boldsymbol{h}_0$ satisfies:

$$\Gamma_{\boldsymbol{p}}(\boldsymbol{y}_l^{\mathrm{std}}) \le \frac{mM}{\sqrt{c-\eta}\,\sigma}\frac{1}{\sqrt{l}}. \tag{30}$$

In particular, even sparse informative signal along $\boldsymbol{p}$ is diluted at rate $O(1/\sqrt{l})$ under uniform residual accumulation.

Proof. For Eq. (28), Assumption 3 bounds the numerator in Eq. (26) by $\sqrt{C}\sigma$, and Lemma 4 bounds the denominator from below by $\sqrt{(c-\eta)l}\sigma$. Dividing these bounds yields Eq. (28).

For Eq. (30), linearity of expectation and Eq. (29) imply:

$$|\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}_l^{\mathrm{std}}\rangle| = \left|\sum_{i=1}^{l} \mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i\rangle\right| = \left|\sum_{i\in S} \mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i\rangle\right| \le mM. \tag{31}$$

By Lemma 4, we also have:

$$\left(\mathbb{E}\|\boldsymbol{y}_l^{\mathrm{std}}\|_2^2\right)^{1/2} \ge \sqrt{(c-\eta)l}\sigma. \tag{32}$$

Substituting into Eq. (27) yields Eq. (30). □

Theorem 6 makes the depth-wise dilution mechanism of standard residuals explicit. Eq. (28) shows that, as the cumulative deviation grows, every depth-stable increment contributes with relative visibility decaying as $O(1/\sqrt{l})$. Eq. (30) extends this conclusion to fixed directions: even when informative signal is confined to a small set S with $|S|$ independent of l, uniform accumulation still drives its directional visibility down at the same rate. Hence depth-wise dilution entails more than aggregate growth and progressively suppresses concentrated informative structure along fixed directions. We therefore analyze normalized depth-wise attention under an explicit signal-plus-noise model to determine whether sparse informative increments can be preserved.

Assumption 7 (Sparse signal plus bounded-correlation noise)**.** Let $\boldsymbol{p} \in \mathbb{R}^d$ be a unit direction along which informative signal is encoded, and let $S \subseteq \{1, \dots, l\}$ be the informative block depths. For $i \in S$, assume:

$$\boldsymbol{v}_i = \mu_i \boldsymbol{p} + \boldsymbol{\xi}_i, \quad \mu_i \geq \mu > 0, \quad \mathbb{E}\boldsymbol{\xi}_i = 0, \quad \mathbb{E}\langle \boldsymbol{p}, \boldsymbol{\xi}_i \rangle = 0. \tag{33}$$

For $i \notin S$, $\boldsymbol{v}_i = \boldsymbol{\xi}_i$ with $\mathbb{E}\boldsymbol{\xi}_i = 0$ and $\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{\xi}_i \rangle = 0$; in particular, the initial value $\boldsymbol{v}_0 = \boldsymbol{h}_0$ is treated as uninformative along $\boldsymbol{p}$ when $0 \notin S$. For all i, $\mathbb{E}\|\boldsymbol{\xi}_i\|_2^2 \leq \tau^2$. There exist constants $0 \leq \gamma < 1$ and $\lambda \in [0, 1]$, all independent of l, $|S|$, μ, and τ, such that for all $i \neq j$,

$$\left|\mathbb{E}\langle \boldsymbol{\xi}_i, \boldsymbol{\xi}_j \rangle\right| \leq \gamma \tau^2 \lambda^{|i-j|}. \tag{34}$$

The parameters γ and λ control only the second-order dependence among noise fluctuations. They are not tied to the signal strength μ or the sparsity level $|S|$. Setting $\lambda = 0$ yields uncorrelated noise across distinct depths, $\lambda < 1$ makes the pairwise bound decay geometrically in the depth separation $|i - j|$, and $\lambda = 1$ reduces to a uniform pairwise bound.

Why is Assumption 7 reasonable? Assumption 7 is a coarse population-level signal-plus-noise model for residual increments, not a literal factorization of every trained $\boldsymbol{v}_i$. It refines the sparse directional condition in Eq. (29) by decomposing $\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i \rangle$ on $i \in S$ into a signed coherent part $\mu_i \geq \mu > 0$ (after orienting $\boldsymbol{p}$) and fluctuations $\boldsymbol{\xi}_i$ with $\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{\xi}_i \rangle = 0$, so analysis along a fixed probe direction $\boldsymbol{p}$, such as a label- or feature-axis in residual space (Lawson et al., 2025), depends only on the component aligned with $\boldsymbol{p}$ while token-specific variation is absorbed into $\boldsymbol{\xi}_i$. The sparse informative set S is consistent with uneven and token-dependent layer contributions (Gromov et al., 2025; Men et al., 2025; Csordás et al., 2025; Lawson et al., 2025). For $i \in S$, $\boldsymbol{v}_i = \mu_i \boldsymbol{p} + \boldsymbol{\xi}_i$ with $\mathbb{E}\boldsymbol{\xi}_i = 0$ does not require Gaussianity or coordinate-wise independence. For $i \notin S$, $\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i \rangle = 0$ is a first-moment condition that allows small per-depth leakage into $\boldsymbol{\xi}_i$ rather than claiming empirically exact zeros. The decay factor $\lambda^{|i-j|}$ in Eq. (34) is a coarse Markovian surrogate for residual recurrence: nearby depths may share more state information, while far-apart depths are nearly uncorrelated in their noise components. The global factor $\gamma < 1$ plays the same role as the weak-alignment constant η in Assumption 3, ruling out fully coherent noise across depth. Because γ and λ are fixed independently of $|S|$ and μ, a depth-independent analysis can treat sparsity through the attention mass placed on S separately from the second-order noise structure controlled by γ and λ in Eqs. (37)-(38). We next quantify, under Assumption 7, how much directional signal a normalized depth readout preserves and how large the remaining aggregated noise can be.

Theorem 8 (Attention residuals preserve sparse informative increments)**.** *Under Assumption 7, let $a_i \geq 0$ with $\sum_{i=0}^{l} a_i = 1$ and set $\boldsymbol{y}_l^{\mathrm{attn}} := \mathcal{A}_l(a)$ as in Eq. (17). If the attention weights assign mass $\rho = \sum_{i \in S} a_i$ to the informative depths, then:*

$$\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}_l^{\mathrm{attn}} \rangle \geq \rho\mu. \tag{35}$$

Define the decay-weighted pairwise mass:

$$\Lambda_\lambda(a) := \sum_{0 \leq i < j \leq l} a_i a_j \, \lambda^{|i-j|}. \tag{36}$$

Then the noise energy satisfies the λ-decay bound:

$$\mathbb{E}\left\| \boldsymbol{y}_l^{\mathrm{attn}} - \sum_{i \in S} a_i \mu_i \boldsymbol{p} \right\|_2^2 \leq \tau^2 \Big(\|a\|_2^2 + 2\gamma \Lambda_\lambda(a) \Big). \tag{37}$$

Since $\lambda^{|i-j|} \leq \lambda$ for all $0 \leq i < j \leq l$ when $\lambda \in [0, 1]$, we have $\Lambda_\lambda(a) \leq \frac{\lambda}{2}\big(1 - \|a\|_2^2\big)$, and hence:

$$\mathbb{E}\left\| \boldsymbol{y}_l^{\mathrm{attn}} - \sum_{i \in S} a_i \mu_i \boldsymbol{p} \right\|_2^2 \leq \tau^2 \big(\gamma\lambda + (1 - \gamma\lambda) \|a\|_2^2 \big). \tag{38}$$

In particular, $\lambda = 0$ yields the uncorrelated-noise envelope $\tau^2 \|a\|_2^2$, while $\lambda = 1$ recovers the uniform-cross-covariance envelope $\tau^2\big(\gamma + (1 - \gamma)\|a\|_2^2\big)$. If $\rho \geq \rho_0 > 0$ and $\|a\|_2 \leq B$ with constants ρ_0, B independent of l, then attention residuals maintain a depth-independent directional signal lower bound while the noise scale is $O(\tau)$ with constant $\sqrt{\gamma\lambda + (1 - \gamma\lambda)B^2}$. Moreover, if

$$\frac{\rho_0 \mu}{\tau} \geq \nu \sqrt{\gamma\lambda + (1 - \gamma\lambda)B^2} \tag{39}$$

for some $\nu > 0$ independent of l, then the directional signal-to-noise scale satisfies:

$$\frac{\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}_l^{\mathrm{attn}} \rangle}{\left(\mathbb{E} \left\| \boldsymbol{y}_l^{\mathrm{attn}} - \sum_{i \in S} a_i \mu_i \boldsymbol{p} \right\|_2^2 \right)^{1/2}} \geq \nu. \tag{40}$$

Sparsity enters through the attention mass ρ_0, while γ and λ enter through the noise envelope in Eqs. (37)-(38).

Proof. By Eq. (17) and Assumption 7, we have:

$$\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}_l^{\mathrm{attn}} \rangle = \sum_{i=0}^{l} a_i \mathbb{E}\langle \boldsymbol{p}, \boldsymbol{v}_i \rangle = \sum_{i \in S} a_i \mu_i \geq \mu \sum_{i \in S} a_i = \rho \mu, \tag{41}$$

which proves Eq. (35).

For the noise bound, decompose the residual output as:

$$\boldsymbol{y}_l^{\mathrm{attn}} - \sum_{i \in S} a_i \mu_i \boldsymbol{p} = \sum_{i=0}^{l} a_i \boldsymbol{\xi}_i. \tag{42}$$

Expanding the squared norm and applying Eq. (34) gives:

$$\mathbb{E} \left\| \sum_{i=0}^{l} a_i \boldsymbol{\xi}_i \right\|_2^2 = \sum_{i=0}^{l} a_i^2 \mathbb{E}\|\boldsymbol{\xi}_i\|_2^2 + 2 \sum_{0 \leq i < j \leq l} a_i a_j \mathbb{E}\langle \boldsymbol{\xi}_i, \boldsymbol{\xi}_j \rangle$$

$$\leq \tau^2 \sum_{i=0}^{l} a_i^2 + 2\gamma\tau^2 \sum_{0 \leq i < j \leq l} a_i a_j \lambda^{|i-j|} = \tau^2 \Big(\|a\|_2^2 + 2\gamma \Lambda_\lambda(a) \Big),$$

which establishes Eq. (37). For $0 \leq i < j \leq l$ we have $\lambda^{|i-j|} \leq \lambda$ because $\lambda \in [0, 1]$, hence:

$$\Lambda_\lambda(a) \leq \lambda \sum_{0 \leq i < j \leq l} a_i a_j = \tfrac{\lambda}{2} \big(1 - \|a\|_2^2 \big), \tag{43}$$

where the last identity uses $\sum_i a_i = 1$. Substituting into Eq. (37) yields Eq. (38). For Eq. (40), Eqs. (35)-(38) with $\rho \geq \rho_0$ and $\|a\|_2 \leq B$ imply $\mathbb{E}\langle \boldsymbol{p}, \boldsymbol{y}_l^{\mathrm{attn}} \rangle \geq \rho_0 \mu$ and $\big(\mathbb{E}\|\boldsymbol{y}_l^{\mathrm{attn}} - \sum_{i \in S} a_i \mu_i \boldsymbol{p}\|_2^2 \big)^{1/2} \leq \tau \sqrt{\gamma\lambda + (1 - \gamma\lambda) B^2}$. Dividing and applying Eq. (39) yields Eq. (40). □

Together, Theorems 6 and 8 isolate a qualitative contrast under Assumptions 3 and 7 when $|S| = m$ does not grow with l. If a normalized depth readout assigns at least constant mass $\rho_0 > 0$ to S, its expected projection onto $\boldsymbol{p}$ is at least $\rho_0 \mu$, and Eq. (39) further yields a depth-independent directional signal-to-noise scale. By contrast, the directional visibility of the same sparse signal in the standard residual stream is $O(1/\sqrt{l})$, regardless of γ, λ, or m. This separation is conditional on informative depths receiving non-negligible weight from the selection mechanism itself. We next make explicit why the choice of residual unit and score calibration can undermine depth-wise selection even under normalized aggregation.

Theorem 9 (Selection bias under heterogeneous residual families)**.** *There exist two residual families, an informative family $\mathcal{G}$ and a distractor family $\mathcal{B}$, such that softmax depth attention over uncalibrated scores assigns almost all mass to $\mathcal{B}$ even though only $\mathcal{G}$ contains informative signal. More precisely, let $\boldsymbol{p}, \boldsymbol{b} \in \mathbb{R}^d$ be orthonormal, let the query be $\boldsymbol{q} = \boldsymbol{b}$, and consider one informative residual value $\boldsymbol{v}_g = \mu \boldsymbol{p}$ with $\mu > 0$ and n distractor values $\boldsymbol{v}_{b,j} = \beta \boldsymbol{b}$ with $\beta > 0$. To isolate score calibration from key normalization, this counterexample uses raw inner-product scores rather than the RMSNorm kernel in Eq. (5); the same bias persists after rescaling both families by a common positive factor. Using the score $s_i = \boldsymbol{q}^\top \boldsymbol{v}_i$ and softmax weights $a_i = \exp(s_i) / \sum_k \exp(s_k)$, the total attention mass on the informative family is:*

$$a_g = \frac{1}{1 + n \exp(\beta)} \tag{44}$$

and the projection of the attention output onto $\boldsymbol{p}$ is:

$$\langle \boldsymbol{p}, \sum_i a_i \boldsymbol{v}_i \rangle = \frac{\mu}{1 + n\exp(\beta)}. \tag{45}$$

Thus, for any $\epsilon > 0$, choosing either sufficiently large n or sufficiently large β makes the informative signal along $\boldsymbol{p}$ smaller than ϵ, despite the presence of an informative residual increment of magnitude μ.

Proof. The informative value has score:

$$s_g = \boldsymbol{q}^\top \boldsymbol{v}_g = \boldsymbol{b}^\top(\mu \boldsymbol{p}) = 0 \tag{46}$$

because $\boldsymbol{p}$ and $\boldsymbol{b}$ are orthonormal. Each distractor value has score:

$$s_{b,j} = \boldsymbol{q}^\top \boldsymbol{v}_{b,j} = \boldsymbol{b}^\top(\beta \boldsymbol{b}) = \beta. \tag{47}$$

Therefore the softmax denominator equals $\exp(0) + n\exp(\beta) = 1 + n\exp(\beta)$, and the informative mass is Eq. (44). The attention output is:

$$a_g \mu \boldsymbol{p} + \sum_{j=1}^{n} a_{b,j} \beta \boldsymbol{b}. \tag{48}$$

Taking the inner product with $\boldsymbol{p}$ removes all distractor terms because $\langle \boldsymbol{p}, \boldsymbol{b} \rangle = 0$, leaving $a_g \mu$, which is Eq. (45). Given $\epsilon > 0$, choose any $\beta > 0$ and $n > \mu/\epsilon$, or choose $n = 1$ and $\beta > \log\big(\max\{0, \mu/\epsilon - 1\}\big)$. Then $\mu/(1 + n\exp(\beta)) < \epsilon$. □

Theorem 10 (Heterogeneous maps couple type selection with information selection)**.** *Let depth indices be partitioned into K residual families $\mathcal{I}_1, \ldots, \mathcal{I}_K$. Assume that a depth-attention score has the form:*

$$s_i = r_i + \omega_{\pi(i)}, \quad i \in \mathcal{I}_{\pi(i)}, \tag{49}$$

where $\pi(i) \in \{1, \ldots, K\}$ indexes the family containing depth i, r_i is the information score within that family, and $\omega_{\pi(i)}$ is a family-dependent calibration offset. Let $S \subseteq \{0, \ldots, l\}$ be the informative depths. Suppose there exist an informative depth $i \in S \cap \mathcal{I}_p$ and an uninformative depth $j \in \mathcal{I}_q \setminus S$ such that $r_i \le r_+$, $r_j \ge r_-$, and $r_+ > r_-$. If the family-dependent offsets obey:

$$\omega_q - \omega_p > r_+ - r_-, \tag{50}$$

then the uninformative depth j receives a larger score than the informative depth i. Consequently, a single attention kernel must learn both the within-family information score r_i and the cross-family calibration offsets ω_k in order to rank informative depths reliably.

Proof. By Eq. (49), the two scores are:

$$s_i = r_i + \omega_p, \quad s_j = r_j + \omega_q. \tag{51}$$

Using $r_i \le r_+$ and $r_j \ge r_-$, their difference is bounded as:

$$s_j - s_i = (r_j - r_i) + (\omega_q - \omega_p) \ge r_- - r_+ + (\omega_q - \omega_p) > 0 \tag{52}$$

by Eq. (50). Thus an uninformative depth can outrank an informative one solely because of family calibration. Reliable ranking across all depths therefore cannot be obtained from information scores alone; the attention mechanism must also estimate or absorb the offsets $\{\omega_k\}_{k=1}^K$. When the residual units alternate between structurally different maps, such as MHA and FFN sublayers, this calibration problem is coupled to the information-selection problem. Bundling MHA and FFN into block-level increments $\mathcal{F}_l$ reduces this a priori family heterogeneity and makes depth attention closer to a single-family selection problem. □

Lemma 4 and Theorem 6 formalize depth-wise dilution under Assumption 3: uniform accumulation grows as $\Theta(l)$, yet fixed increments and sparse directional signal have relative visibility $O(1/\sqrt{l})$. Under Assumption 7, Theorem 8 shows that a normalized readout can preserve informative projections and achieve depth-independent directional SNR when informative depths receive non-negligible weight, whereas Theorems 9 and 10 show that heterogeneous residual units and

uncalibrated scores can undermine such selection and motivate block-level increments $\mathcal{F}_l$ over alternating MHA/FFN sublayers. Accordingly, LLARA (Section 3) instantiates this design with a frozen block recurrence, block-level residual values $\{\boldsymbol{v}_l\}_{l=0}^{L}$ from Eq. (7), and an output-only correction:

$$\Delta \boldsymbol{h} = \sum_{i=0}^{L} a_i \boldsymbol{v}_i, \quad a_i = \frac{w_i}{\sum_{j=0}^{L} w_j}, \quad w_i = \varphi(\boldsymbol{q})^\top \boldsymbol{k}_i > 0 \tag{53}$$

from the low-rank linear-attention readout in Eqs. (10)-(13), applied via Eq. (8). When the learned readout places non-negligible mass on informative block residuals, the correction $\Delta \boldsymbol{h}$ in Eq. (53) satisfies the signal-preservation and SNR guarantees of Theorem 8, including Eq. (40) under Eq. (39). These guarantees are stated for the normalized depth readout $\Delta \boldsymbol{h}$ itself and do not automatically transfer to the relative-visibility metric of Theorem 6 applied to the full adapted state $\tilde{\boldsymbol{h}}_L = \boldsymbol{h}_L + \Delta \boldsymbol{h}$, because $\boldsymbol{h}_L$ retains the $\Theta(\sqrt{L})$-scale cumulative deviation analyzed in Lemma 4.

Proposition 11 (Effect of output normalization on the LLARA correction)**.** *Fix block depth L and suppose $\|\Delta \boldsymbol{h}\|_2 \le V$. Let $\boldsymbol{h}_L$ be the frozen-backbone final state from Eq. (7) and define $\tilde{\boldsymbol{h}}_L = \boldsymbol{h}_L + \Delta \boldsymbol{h}$ as in Eq. (8). Assume $\mathrm{Norm}_{\mathrm{out}}$ is the RMSNorm map in Eq. (18). If $\|\boldsymbol{h}_L\|_2 \ge \rho_h \sqrt{L}$ for some constant $\rho_h > 0$ independent of L, and L is large enough that $\rho_h \sqrt{L} > V$, then*

$$\left\| \mathrm{Norm}_{\mathrm{out}}(\tilde{\boldsymbol{h}}_L) - \mathrm{Norm}_{\mathrm{out}}(\boldsymbol{h}_L) \right\|_2 \le \frac{\sqrt{d}\, V}{\rho_h \sqrt{L} - V}. \tag{54}$$

In particular, for any unit direction $\boldsymbol{p} \in \mathbb{R}^d$,

$$\left| \left\langle \boldsymbol{p}, \mathrm{Norm}_{\mathrm{out}}(\tilde{\boldsymbol{h}}_L) - \mathrm{Norm}_{\mathrm{out}}(\boldsymbol{h}_L) \right\rangle \right| \le \frac{\sqrt{d}\, V}{\rho_h \sqrt{L} - V}. \tag{55}$$

Both right-hand sides equal $\big(\sqrt{d}\, V/(\rho_h \sqrt{L})\big)\big(1 + O(L^{-1/2})\big)$, so the normalized feature change induced by $\Delta \boldsymbol{h}$ is $O(1/\sqrt{L})$ relative to the backbone scale. The precondition $\|\boldsymbol{h}_L\|_2 = \Omega(\sqrt{L})$ is discussed in Remark 12.

Proof. Write $\boldsymbol{h} = \boldsymbol{h}_L$, $\Delta = \Delta \boldsymbol{h}$, and $f(\boldsymbol{v}) = \mathrm{RMSNorm}(\boldsymbol{v}) = \boldsymbol{v}/\mathrm{rms}(\boldsymbol{v})$. Differentiating Eq. (18) gives the Jacobian:

$$Df(\boldsymbol{v}) = \frac{1}{\mathrm{rms}(\boldsymbol{v})} \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}\boldsymbol{v}^\top}{d\, \mathrm{rms}(\boldsymbol{v})^2} \right). \tag{56}$$

The rank-one matrix $\boldsymbol{v}\boldsymbol{v}^\top/(d\,\mathrm{rms}(\boldsymbol{v})^2)$ is positive semidefinite with single non-zero eigenvalue $\|\boldsymbol{v}\|_2^2/(\|\boldsymbol{v}\|_2^2 + d\delta) \in [0, 1)$, so all eigenvalues of the bracket lie in $(0, 1]$ and $\|Df(\boldsymbol{v})\|_2 \le 1/\mathrm{rms}(\boldsymbol{v})$.

Eqs. (54)-(55) compare f at two distinct points rather than along an infinitesimal variation, so the endpoint Jacobian norm $1/\mathrm{rms}(\boldsymbol{h})$ does not bound the difference by itself: $\mathrm{rms}(\boldsymbol{h} + t\Delta)$ can be strictly smaller than $\mathrm{rms}(\boldsymbol{h})$ when Δ has a component opposite to $\boldsymbol{h}$. We therefore apply the mean value inequality along the segment $\boldsymbol{h}_t = \boldsymbol{h} + t\Delta$, $t \in [0, 1]$, on which f is continuously differentiable because $\mathrm{rms} \ge \sqrt{\delta} > 0$:

$$\big\| f(\boldsymbol{h}+\Delta) - f(\boldsymbol{h}) \big\|_2 = \left\| \int_0^1 Df(\boldsymbol{h}_t)\, \Delta \, \mathrm{d}t \right\|_2 \le \left(\sup_{t \in [0,1]} \big\| Df(\boldsymbol{h}_t) \big\|_2 \right) \|\Delta\|_2 = \frac{\|\Delta\|_2}{\inf_{t \in [0,1]} \mathrm{rms}(\boldsymbol{h}_t)}. \tag{57}$$

For every $t \in [0, 1]$, the triangle inequality together with $\mathrm{rms}(\boldsymbol{v}) = \sqrt{\|\boldsymbol{v}\|_2^2/d + \delta} \ge \|\boldsymbol{v}\|_2/\sqrt{d}$ gives:

$$\mathrm{rms}(\boldsymbol{h}_t) \ge \frac{\|\boldsymbol{h}_t\|_2}{\sqrt{d}} \ge \frac{\|\boldsymbol{h}\|_2 - \|\Delta\|_2}{\sqrt{d}} \ge \frac{\rho_h \sqrt{L} - V}{\sqrt{d}} > 0, \tag{58}$$

using $\|\boldsymbol{h}\|_2 \ge \rho_h \sqrt{L}$, $\|\Delta\|_2 \le V$, and $\rho_h \sqrt{L} > V$. Substituting this segment-wide lower bound into Eq. (57) yields Eq. (54), and Eq. (55) follows by Cauchy–Schwarz. □

Remark 12 (On the precondition $\|\boldsymbol{h}_L\|_2 = \Omega(\sqrt{L})$). Proposition 11 is a deterministic statement conditional on the realized norm $\|\boldsymbol{h}_L\|_2$, and Lemma 4 does not deliver its precondition directly, for

two distinct reasons. First, the lemma controls the cumulative deviation $\boldsymbol{h}_L - \boldsymbol{h}_0$ rather than $\boldsymbol{h}_L$ itself. This gap is harmless: Minkowski's inequality in L^2 and Lemma 4 give

$$\big(\mathbb{E}\|\boldsymbol{h}_L\|_2^2\big)^{1/2} \geq \big(\mathbb{E}\|\boldsymbol{h}_L - \boldsymbol{h}_0\|_2^2\big)^{1/2} - \big(\mathbb{E}\|\boldsymbol{h}_0\|_2^2\big)^{1/2} \geq \sqrt{(c-\eta)L}\,\sigma - \big(\mathbb{E}\|\boldsymbol{h}_0\|_2^2\big)^{1/2}, \tag{59}$$

and $\mathbb{E}\|\boldsymbol{h}_0\|_2^2$ is the token-embedding scale, which does not grow with L. Second, and less innocuously, Eq. (59) constrains only a second moment, whereas the precondition is pointwise: a nonnegative random variable with large mean square can still be small with high probability, so an anti-concentration condition is required. If the fourth moment is comparable to the squared second moment, that is, $\mathbb{E}\|\boldsymbol{h}_L\|_2^4 \leq K_4\big(\mathbb{E}\|\boldsymbol{h}_L\|_2^2\big)^2$ for a constant K_4 independent of L, then the Paley–Zygmund inequality applied to $Z = \|\boldsymbol{h}_L\|_2^2$ gives, for every $\theta \in (0,1)$:

$$\mathbb{P}\Big(\|\boldsymbol{h}_L\|_2^2 > \theta\,\mathbb{E}\|\boldsymbol{h}_L\|_2^2\Big) \geq \frac{(1-\theta)^2}{K_4}. \tag{60}$$

For L large enough that Eq. (59) yields $\mathbb{E}\|\boldsymbol{h}_L\|_2^2 \geq \frac{1}{2}(c-\eta)L\sigma^2$, the precondition then holds with $\rho_h = \sigma\sqrt{\theta(c-\eta)/2}$ on an event of probability at least $(1-\theta)^2/K_4$. We therefore treat $\|\boldsymbol{h}_L\|_2 = \Omega(\sqrt{L})$ as an additional regularity condition on the frozen trajectory, consistent with Lemma 4 at the level of second moments, rather than as a consequence of it.

Thus LLARA separates two roles at the output: $\Delta\boldsymbol{h}$ performs selective, depth-normalized recomposition of the realized increments, while $\boldsymbol{h}_L + \Delta\boldsymbol{h}$ preserves the pretrained block interface and $\mathrm{Norm}_{\mathrm{out}}$ maps the adapted state into the feature scale expected by the frozen language-modeling head. The anti-dilution analysis therefore applies to $\Delta\boldsymbol{h}$, and Proposition 11 quantifies how much of that correction survives in the normalized features consumed by $\boldsymbol{W}_{\mathrm{lm}}$.

D Background on Linear Attention

Self-attention maps an input sequence $x \in \mathbb{R}^{N\times F}$ to queries, keys, and values via $Q = xW_Q \in \mathbb{R}^{N\times D}$, $K = xW_K \in \mathbb{R}^{N\times D}$, and $V = xW_V \in \mathbb{R}^{N\times M}$. For each position i, the attention output is a normalized similarity-weighted average of values:

$$V_i' = \frac{\sum_{j=1}^N \mathrm{sim}(Q_i, K_j)\,V_j}{\sum_{j=1}^N \mathrm{sim}(Q_i, K_j)}, \tag{61}$$

where $\mathrm{sim}(\cdot,\cdot) \geq 0$ ensures that V_i' is a convex combination of $\{V_j\}_{j=1}^N$ whenever the denominator is nonzero. Softmax attention corresponds to $\mathrm{sim}(q,k) = \exp(q^\top k/\sqrt{D})$. Softmax attention must materialize or implicitly revisit all N^2 pairwise interactions, yielding quadratic cost in the sequence length and requiring an $N \times N$ attention map in the standard implementation.

Linear attention (Katharopoulos et al., 2020) replaces the exponential softmax kernel by a factorizable positive kernel:

$$\mathrm{sim}(q,k) = \varphi(q)^\top \varphi(k), \tag{62}$$

where $\varphi : \mathbb{R}^D \to \mathbb{R}^C_{\geq 0}$ is applied row-wise to Q and K. Rather than constituting an exact finite-dimensional rewriting of softmax attention, this formulation defines a different normalized positive kernel whose algebraic structure admits a more efficient evaluation. Substituting Eq. (62) into Eq. (61) and using associativity yields:

$$V_i' = \frac{\varphi(Q_i)^\top \sum_{j=1}^N \varphi(K_j)\,V_j^\top}{\varphi(Q_i)^\top \sum_{j=1}^N \varphi(K_j)}, \tag{63}$$

Equivalently, in vectorized form:

$$(\varphi(Q)\varphi(K)^\top)V = \varphi(Q)(\varphi(K)^\top V). \tag{64}$$

The $N \times N$ attention matrix is therefore never formed, and the sufficient statistics are defined as:

$$\mathbf{S} = \sum_{j=1}^N \varphi(K_j)V_j^\top \in \mathbb{R}^{C\times M}, \quad \mathbf{z} = \sum_{j=1}^N \varphi(K_j) \in \mathbb{R}^C, \tag{65}$$

and the query readout is given by:

$$V_i' = \frac{\varphi(Q_i)^\top \mathbf{S}}{\varphi(Q_i)^\top \mathbf{z} + \varepsilon}, \quad \varepsilon > 0. \tag{66}$$

The constant ε stabilizes the denominator when the accumulated normalization mass is numerically tiny. The dominant cost is $\mathcal{O}(NCM)$ arithmetic operations and $\mathcal{O}(CM + C)$ state memory, linear in N when the feature dimension C and value dimension M are fixed.

For autoregressive or streaming computation, the same construction is applied in prefix form:

$$\mathbf{S}_i = \sum_{j=1}^{i} \varphi(K_j) V_j^\top, \quad \mathbf{z}_i = \sum_{j=1}^{i} \varphi(K_j), \quad V_i' = \frac{\varphi(Q_i)^\top \mathbf{S}_i}{\varphi(Q_i)^\top \mathbf{z}_i + \varepsilon}. \tag{67}$$

Thus keys and values can be consumed online through the constant-size updates $\mathbf{S}_i = \mathbf{S}_{i-1} + \varphi(K_i)V_i^\top$ and $\mathbf{z}_i = \mathbf{z}_{i-1} + \varphi(K_i)$. The same prefix construction applies along decoder depth in LLARA: replacing the sequence index i with the depth position l compresses each token's depth history into fixed-size online accumulators rather than an explicit residual list. Accordingly, linear attention enables normalized, non-negative, input-dependent mixing over depth with state memory independent of the number of decoder blocks. A commonly used positive feature map is $\varphi(x) = \mathrm{ELU}(x) + 1$ (Clevert et al., 2016), which preserves strict coordinate positivity while maintaining gradients on negative inputs, and LLARA adopts this map for its low-rank depth keys and the learnable query used in the final output readout.

E Details of LoRA

LoRA (Hu et al., 2022) is a parameter-efficient fine-tuning method that adapts a pretrained model by learning a low-rank update to selected linear maps while keeping the original weights frozen. Consider a pretrained linear transformation with frozen weight $\boldsymbol{W}_0 \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ and input $\boldsymbol{x} \in \mathbb{R}^{d_{\mathrm{in}}}$. Instead of optimizing a full dense update $\Delta \boldsymbol{W}$ with $d_{\mathrm{out}} d_{\mathrm{in}}$ trainable parameters, LoRA parameterizes the update as:

$$\Delta \boldsymbol{W} = \frac{\alpha}{r_{\mathrm{LoRA}}} \boldsymbol{B}\boldsymbol{A}, \quad \boldsymbol{A} \in \mathbb{R}^{r_{\mathrm{LoRA}} \times d_{\mathrm{in}}}, \quad \boldsymbol{B} \in \mathbb{R}^{d_{\mathrm{out}} \times r_{\mathrm{LoRA}}}, \tag{68}$$

where $r_{\mathrm{LoRA}} \ll \min(d_{\mathrm{in}}, d_{\mathrm{out}})$ is the LoRA rank and α is a scaling hyperparameter. The adapted forward computation is:

$$\boldsymbol{y} = \boldsymbol{W}_0 \boldsymbol{x} + \frac{\alpha}{r_{\mathrm{LoRA}}} \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}. \tag{69}$$

Thus the number of trainable parameters for this layer is reduced from $d_{\mathrm{out}} d_{\mathrm{in}}$ to $r_{\mathrm{LoRA}}(d_{\mathrm{in}} + d_{\mathrm{out}})$. In the common initialization, $\boldsymbol{A}$ is randomly initialized and $\boldsymbol{B}$ is initialized to zero, giving $\Delta \boldsymbol{W} = \boldsymbol{0}$ at the beginning of training. Consequently, the initial adapted model computes exactly the same function as the frozen pretrained model, and task-specific behavior is introduced only through the learned low-rank factors.

After training, the LoRA update for a linear map can be merged into an effective static weight:

$$\boldsymbol{W}_{\mathrm{eff}} = \boldsymbol{W}_0 + \frac{\alpha}{r_{\mathrm{LoRA}}} \boldsymbol{B}\boldsymbol{A}. \tag{70}$$

This mergeability follows from the fact that LoRA modifies only the affine weight used by the selected linear transformation. It is therefore a parameter adaptation mechanism inside existing Transformer blocks rather than a new recurrent or stateful architectural pathway.

LoRA is compatible with LLARA because the two mechanisms act on complementary parts of the computation graph. In the notation of Eq. (6), LoRA changes the within-block maps appearing inside $\mathcal{F}_l$ by replacing selected frozen linear weights with their effective adapted weights from Eq. (70). The block recurrence itself is still executed in the usual order, producing the realized residual values $\{\boldsymbol{v}_l\}_{l=0}^{L}$ defined in Eq. (7). LLARA then operates outside the block: it observes these realized increments, accumulates the low-rank linear-attention state over depth, and adds the output-level correction $\Delta \boldsymbol{h}$ in Eq. (8). Equivalently, LoRA adapts the local transformation that produces each residual increment, while LLARA adapts the global depth-wise composition of those increments.

Algorithm 1 LLARA decoder-stack forward pass.

Require: Decoder blocks $\{\mathcal{B}_l\}_{l=1}^{L}$, hidden state $\boldsymbol{h}_0$, projections $\{\boldsymbol{A}_l\}_{l=0}^{L}$, query $\boldsymbol{q}$, stabilizer ε
Ensure: Adapted final hidden state $\tilde{\boldsymbol{h}}_L$

$\boldsymbol{S} \leftarrow \boldsymbol{0} \in \mathbb{R}^{r\times d}$, $\boldsymbol{z} \leftarrow \boldsymbol{0} \in \mathbb{R}^{r}$
$\boldsymbol{k}_0 \leftarrow \varphi(\boldsymbol{A}_0\,\mathrm{RMSNorm}(\boldsymbol{h}_0))$
$\boldsymbol{S} \leftarrow \boldsymbol{S} + \boldsymbol{k}_0\boldsymbol{h}_0^{\top}$, $\boldsymbol{z} \leftarrow \boldsymbol{z} + \boldsymbol{k}_0$
for $l = 1, \ldots, L$ **do**
    $\boldsymbol{h}_l \leftarrow \mathcal{B}_l(\boldsymbol{h}_{l-1})$ ▷ run the original block without changing its input
    $\boldsymbol{v}_l \leftarrow \boldsymbol{h}_l - \boldsymbol{h}_{l-1}$
    $\boldsymbol{k}_l \leftarrow \varphi(\boldsymbol{A}_l\,\mathrm{RMSNorm}(\boldsymbol{v}_l))$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} + \boldsymbol{k}_l\boldsymbol{v}_l^{\top}$, $\boldsymbol{z} \leftarrow \boldsymbol{z} + \boldsymbol{k}_l$
end for
$\Delta\boldsymbol{h} \leftarrow \dfrac{\boldsymbol{S}^{\top}\varphi(\boldsymbol{q})}{\varphi(\boldsymbol{q})^{\top}\boldsymbol{z} + \varepsilon}$
$\tilde{\boldsymbol{h}}_L \leftarrow \boldsymbol{h}_L + \Delta\boldsymbol{h}$
return $\tilde{\boldsymbol{h}}_L$

The resulting trainable parameter set Θ_{train} is parameter-disjoint: LoRA owns the low-rank factors in Θ_{LoRA}, whereas LLARA owns the depth-key projections $\{\boldsymbol{A}_l\}_{l=0}^{L}$ and the readout query $\boldsymbol{q}$. During joint training, gradients may couple through the loss because LLARA reads the trajectory produced by the LoRA-adapted backbone, but there is no parameter sharing or incompatible overwrite between the two adapters. This is also why the two inference behaviors differ: LoRA's learned matrices can be folded into static weights layer by layer, while LLARA's learned correction depends on the input-specific hidden trajectory and the online depth state and must remain attached as a residual adaptation pathway.

F Implementation Details of LLARA

This section maps the construction in Section 3 to the PEFT block-wrapper design used in our codebase. Algorithm 1 and Figure 5 specify the forward logic. The prose below explains the architectural choices that are not visible from the equations alone.

F.1 Architecture and Injection

LLARA is implemented as a PEFT tuner that wraps each matched decoder block with a thin outer module. This placement at the block boundary is deliberate. LoRA adapters modify within-block linear maps, whereas LLARA sits outside the block and reads the *realized* residual trajectory produced by the frozen recurrence in Eq. (6). The wrapper forwards `hidden_states`, `*args`, and `**kwargs` to the inner block unchanged. The pretrained block interface is therefore preserved, including attention masks, position identifiers, and key-value caches.

Injection proceeds in two phases. In the pre-injection phase, a hook allocates one shared `LLARADepthState` for the entire decoder stack. Every wrapper holds a reference to this object rather than registering $(\boldsymbol{S}, \boldsymbol{z})$ as model parameters or persistent buffers. The layer-0 wrapper resets the state at the start of each forward pass. During module traversal, each target block is wrapped, its layer index is parsed from the trailing digits of the module key, and a per-layer key projection is attached. The layer-0 wrapper additionally owns a separate input projection for $\boldsymbol{v}_0 = \boldsymbol{h}_0$. This projection is stored as `llara_key_proj_in` and is kept separate from the residual keys in `llara_key_proj`. All key projections and the LLARA query are initialized to zero, so $\boldsymbol{k}_l = \boldsymbol{u} = \varphi(\boldsymbol{0}) = \boldsymbol{1}$ at the start of training. The initial readout therefore applies uniform depth weights before selectivity emerges during optimization. A post-injection hook sets `is_last_layer` on the wrapper with the maximum parsed index and attaches the query $\boldsymbol{q}$ only on that wrapper. This matches the output-only readout in Eq. (13). If `target_modules` is not specified manually, the tuner selects a decoder-layer regex from `model_type` for common backbones such as LLaMA, Mistral, Gemma, and Qwen. Unsupported architectures require an explicit pattern.

```python
class LLARADecoderLayerWrapper(nn.Module):  # + LLARALayer; __init__ omitted
    def forward(self, hidden_states, *args, **kwargs):
        if self.disable_adapters or not self.active_adapters:
            return self.inner(hidden_states, *args, **kwargs)

        adapter = self.active_adapters[0]
        state = self._llara_depth_state
        kp = self.llara_key_proj[adapter]

        if self.layer_idx == 0:
            state.reset()
            if adapter not in self.llara_key_proj_in:
                raise RuntimeError("missing llara_key_proj_in at layer 0")
            state.update(hidden_states, self.llara_key_proj_in[adapter])

        output = self.inner(hidden_states, *args, **kwargs)
        new_v = output[0] if isinstance(output, (tuple, list)) else output
        state.update(new_v - hidden_states, kp)

        if self.is_last_layer:
            q_feat = phi(self.llara_query[adapter])
            delta = state.mix(q_feat).to(new_v.dtype)
            if isinstance(output, (tuple, list)):
                return (output[0] + delta,) + tuple(output[1:])
            return output + delta
        return output

class LLARADepthState:
    def __init__(self, eps=1e-6):
        self.eps = eps
        self.S, self.z = None, None

    def reset(self):
        self.S, self.z = None, None

    def update(self, v, key_proj):
        k_raw = rms_norm(v)
        k_feat = phi(key_proj(k_raw.to(key_proj.weight.dtype)))
        if v.dtype in (torch.float16, torch.bfloat16):
            k_feat_acc, v_acc = k_feat.float(), v.float()
        else:
            k_feat_acc, v_acc = k_feat, v
        outer = k_feat_acc.unsqueeze(-1) * v_acc.unsqueeze(-2)
        if self.S is None:
            self.S, self.z = outer, k_feat_acc
        else:
            self.S.add_(outer); self.z.add_(k_feat_acc)

    def mix(self, q_feat):
        if self.S is None:
            raise RuntimeError("mix() called before any update()")
        with torch.autocast(q_feat.device.type, enabled=False):
            num = torch.einsum("j,btjd->btd", q_feat, self.S)
            den = torch.einsum("j,btj->bt", q_feat, self.z).unsqueeze(-1) + self.eps
            return num / den
```

Figure 5: PyTorch-style pseudocode for the implemented LLARA wrapper and online depth state.

When LLARA is trained jointly with LoRA, the causal order is within-block adaptation followed by cross-block aggregation. LoRA changes $\mathcal{F}_l$ inside each block, and the wrapper observes the resulting realized increments $\{\boldsymbol{v}_l\}$ without sharing parameters with LoRA.

F.2 Forward State and PyTorch-Style Pseudocode

The runtime depth state consists of two tensors, `S` and `z`, held in a shared `LLARADepthState` object. They correspond to $\boldsymbol{S}$ and $\boldsymbol{z}$ in Algorithm 1 but are neither model parameters nor persistent

buffers. Each forward pass begins with a reset at layer 0, followed by depth-ordered updates at every wrapped block and a readout at the last block. Figure 5 shows how the wrapper dispatches these updates and how `mix` evaluates the normalized readout in floating point.

F.3 Training Interface and Design Limits

LLARA is exposed through the standard PEFT adapter interface with the prefix `llara_`. When trained jointly with LoRA, the two tuners remain parameter-disjoint. LoRA owns the within-block low-rank matrices, while LLARA owns $\{\boldsymbol{A}_l\}_{l=0}^{L}$ and $\boldsymbol{q}$ under the `llara_` namespace. The learned correction depends on the full hidden trajectory and the online depth state. It cannot therefore be folded into a static backbone weight update. Layer-level merge and unmerge are unsupported, and unloading the adapter removes the wrappers and discards the trained LLARA behavior. For inference, the adapter should remain attached through the PEFT model interface.

Mixed-precision training promotes `float16` and `bfloat16` keys and values to `float32` before accumulating $\boldsymbol{S}$. The final readout is evaluated with autocast disabled, and $\Delta\boldsymbol{h}$ is then cast back to the hidden-state dtype (Figure 5). These choices stabilize the normalized denominator in Eq. (13) while preserving the dtype contract expected by the surrounding transformer stack.

F.4 Checkpointing Compatibility

The depth state is reconstructed by deterministic, depth-ordered wrapper execution rather than stored as a persistent buffer or materialized depth cache. Activation checkpointing is therefore compatible whenever recomputation replays wrapped blocks in the same order. The layer-0 reset must precede subsequent updates. The wrapper does not reinterpret model-specific arguments passed through to the inner block. Key-value caching and other backbone-side optimizations therefore remain governed by the underlying pretrained implementation.

G Gradient Analysis of LLARA

This appendix supplies the block-level setup, gradient identities, and proofs for Theorem 1 in §3.5. Throughout, we fix a single token, omit batch indices, and analyze SGD gradients. We write $\boldsymbol{g} = \nabla_{\boldsymbol{h}}\mathcal{L} \in \mathbb{R}^d$ for the upstream gradient with respect to a hidden vector $\boldsymbol{h}$.

G.1 Setup and Notation

Loss chain. LLARA training minimizes a scalar loss $\mathcal{L}$ that depends on the adapted final hidden state $\tilde{\boldsymbol{h}}_L$ through the language-modeling head in Eq. (14). By the chain rule,

$$\nabla_{\Delta\boldsymbol{h}}\mathcal{L} = \nabla_{\tilde{\boldsymbol{h}}_L}\mathcal{L}, \tag{71}$$

since $\tilde{\boldsymbol{h}}_L = \boldsymbol{h}_L + \Delta\boldsymbol{h}$ and $\boldsymbol{h}_L$ is treated as constant with respect to LLARA parameters during post-training.

Block-level residual values. The block-level trajectory is defined in Eq. (7):

$$\boldsymbol{v}_0 = \boldsymbol{h}_0, \quad \boldsymbol{v}_l = \boldsymbol{h}_l - \boldsymbol{h}_{l-1} = \mathcal{F}_l(\boldsymbol{h}_{l-1}), \quad l = 1, \ldots, L. \tag{72}$$

Block-level AttnRes. For comparison with LLARA, we instantiate AttnRes at the decoder-block level. For each trainable query block $l \in \{1, \ldots, L\}$, define normalized keys $\tilde{\boldsymbol{k}}_i = \mathrm{RMSNorm}(\boldsymbol{v}_i)$, scores $s_{i\to l} = \boldsymbol{q}_l^\top \tilde{\boldsymbol{k}}_i$, softmax weights

$$\alpha_{i\to l} = \frac{\exp(s_{i\to l})}{\sum_{j=0}^{l}\exp(s_{j\to l})}, \quad i = 0, \ldots, l, \tag{73}$$

and the depth-attention aggregate

$$\boldsymbol{h}_l = \sum_{i=0}^{l} \alpha_{i\to l}\boldsymbol{v}_i. \tag{74}$$

The trainable AttnRes parameters are $\{\boldsymbol{q}_l\}_{l=1}^{L} \subset \mathbb{R}^d$. Residual increments satisfy $\boldsymbol{v}_{l+1} = \mathcal{F}_{l+1}(\boldsymbol{h}_l)$, so the AttnRes aggregate $\boldsymbol{h}_l$ in Eq. (74) enters all subsequent blocks and aggregates. In AttnRes, $\boldsymbol{h}_l$ denotes the depth-attention aggregate rather than the backbone hidden state in Eq. (7). **The block-level AttnRes used here is not the Block AttnRes of (Kimi Team, 2026a).** That variant partitions L layers into $N \ll L$ groups, reduces each group by standard residual summation, and applies softmax attention only over the N block summaries to cut memory and communication from $O(Ld)$ to $O(Nd)$. By contrast, we instantiate the full AttnRes recurrence of Eq. (4) at decoder-block granularity: each increment $\mathcal{F}_l$ remains a separate residual value, and later blocks attend over the entire block-residual trajectory rather than over a handful of intra-block sums.

LLARA readout. Define $\boldsymbol{x}_l = \mathrm{RMSNorm}(\boldsymbol{v}_l)$, $\boldsymbol{k}_l = \varphi(\boldsymbol{A}_l \boldsymbol{x}_l) \in \mathbb{R}^r_{>0}$, $\boldsymbol{u} = \varphi(\boldsymbol{q}) \in \mathbb{R}^r_{>0}$, and the sufficient statistics

$$\boldsymbol{S}_L = \sum_{i=0}^{L} \boldsymbol{k}_i \boldsymbol{v}_i^\top \in \mathbb{R}^{r \times d}, \quad \boldsymbol{z}_L = \sum_{i=0}^{L} \boldsymbol{k}_i \in \mathbb{R}^r. \tag{75}$$

The normalized readout (ignoring ε for the theoretical analysis, as in Appendix C) is

$$\Delta \boldsymbol{h} = \frac{\boldsymbol{S}_L^\top \boldsymbol{u}}{\boldsymbol{u}^\top \boldsymbol{z}_L} = \frac{\sum_{i=0}^{L} w_i \boldsymbol{v}_i}{\sum_{i=0}^{L} w_i}, \quad w_i = \boldsymbol{u}^\top \boldsymbol{k}_i, \tag{76}$$

matching Eq. (13) with $\boldsymbol{u} = \varphi(\boldsymbol{q})$.

Remark 13 (Stabilizer ε). The implementation adds $\varepsilon > 0$ to the denominator in Eq. (13). This affects gradients only through the scalar denominator: all formulas below remain valid after replacing $D = \boldsymbol{u}^\top \boldsymbol{z}_L$ by $D = \boldsymbol{u}^\top \boldsymbol{z}_L + \varepsilon$ and using $\Delta \boldsymbol{h} = N/D$.

Assumption 14 (Effective recurrent sensitivity for AttnRes)**.** In the block-level AttnRes recurrence of Eqs. (73) and (74), let $\boldsymbol{h}_m$ denote the depth-attention aggregate rather than the backbone hidden state in Eq. (7). The AttnRes state at depth m is the concatenated vector $\boldsymbol{\zeta}_m = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_m, \boldsymbol{h}_m) \in \mathbb{R}^{(m+2)d}$, with initial state $\boldsymbol{\zeta}_0 = (\boldsymbol{v}_0, \boldsymbol{h}_0)$ and $\boldsymbol{h}_0 = \boldsymbol{v}_0$. For each $m = 1, \ldots, L$, define the transition $\boldsymbol{\zeta}_m = \mathcal{T}_m(\boldsymbol{\zeta}_{m-1}; \boldsymbol{q}_m)$ by $\boldsymbol{v}_m = \mathcal{F}_m(\boldsymbol{h}_{m-1})$, normalized keys $\tilde{\boldsymbol{k}}_i = \mathrm{RMSNorm}(\boldsymbol{v}_i)$, and the softmax weights and aggregate in Eqs. (73) and (74), holding $\boldsymbol{q}_m$ fixed when differentiating with respect to $\boldsymbol{\zeta}_{m-1}$. All vector norms are Euclidean norms on concatenated state components, and $\|\cdot\|_2$ on Jacobians denotes the induced operator norm. Along the realized trajectory at the current parameter values, there exists a finite constant $L_{\mathrm{AttnRes}} > 0$ such that:

$$\left\| \frac{\partial \mathcal{T}_m}{\partial \boldsymbol{\zeta}_{m-1}} \right\|_2 \leq L_{\mathrm{AttnRes}}, \quad m = 1, \ldots, L. \tag{77}$$

Why is Assumption 14 reasonable? Assumption 14 is used only in Theorem 1(i) and plays no role in the LLARA bounds in item (ii). It bounds the sensitivity of the AttnRes state map $\mathcal{T}_m$, rather than the frozen backbone block $\mathcal{F}_m$ alone. This distinction matters because later depth aggregates depend on earlier state components both through the residual values $\boldsymbol{v}_i$ and through the normalized keys $\tilde{\boldsymbol{k}}_i = \mathrm{RMSNorm}(\boldsymbol{v}_i)$ in the softmax scores. The constant L_{AttnRes} therefore aggregates the block Jacobian of $\mathcal{F}_m$, the RMSNorm derivative, the softmax Jacobian, and the finite fan-in of the depth aggregate on the realized trajectory. Together with the AttnRes envelope conditions in Assumption 15, the residual and key norms are bounded on a fixed forward pass at the current parameters. On this set, the frozen blocks $\mathcal{F}_m$, RMSNorm, and softmax form smooth compositions, so each transition $\mathcal{T}_m$ admits a finite local Lipschitz constant and one may take $L_{\mathrm{AttnRes}} := \max_{m=1,\ldots,L} \|\partial \mathcal{T}_m / \partial \boldsymbol{\zeta}_{m-1}\|_2$ along the trajectory. Reverse-mode backpropagation through $\mathcal{T}_{l+1}, \ldots, \mathcal{T}_L$ then recovers the standard adjoint-product mechanism familiar from depth-recurrent and recurrent-network gradient analysis. The assumption requires only such a finite local bound and does not impose contractivity or $L_{\mathrm{AttnRes}} < 1$. A larger constant loosens only the AttnRes side of the comparison, because the query bound scales with $\prod_{m=l+1}^{L} L_{\mathrm{AttnRes}}$. The resulting contrast is therefore structural rather than tied to a sharp numeric estimate. AttnRes query gradients can inherit depth products along the recurrence, while LLARA architectural gradients do not, regardless of how tight the local Lipschitz envelope is.

Assumption 15 (Bounded gradients and readout quantities)**.** Fix a token. For each mechanism, consider the block-level residuals $\{\boldsymbol{v}_l\}_{l=0}^{L}$ along its realized forward trajectory at the current parameter

values. There exist constants $G, V, \kappa, d_{\min} > 0$ and $B_\varphi \geq 1$ such that, for AttnRes,

$$\|\nabla_{\boldsymbol{\zeta}_L} \mathcal{L}\|_2 \leq G, \quad \|\boldsymbol{v}_l\|_2 \leq V \ \ (l = 0, \ldots, L), \quad \|\tilde{\boldsymbol{k}}_i\|_2 \leq \sqrt{d} \ \ (i = 0, \ldots, L), \tag{78}$$

and for LLARA with $\boldsymbol{u} = \varphi(\boldsymbol{q})$, $\boldsymbol{k}_l = \varphi(\boldsymbol{A}_l \boldsymbol{x}_l)$, $\boldsymbol{x}_l = \mathrm{RMSNorm}(\boldsymbol{v}_l)$ as in Eq. (18), and $D = \boldsymbol{u}^\top \boldsymbol{z}_L$,

$$\begin{aligned} &\|\nabla_{\Delta \boldsymbol{h}} \mathcal{L}\|_2 \leq G, \quad \|\boldsymbol{v}_l\|_2 \leq V \ \ (l = 0, \ldots, L), \quad \|\boldsymbol{u}\|_2 \leq \kappa, \\ &\|\boldsymbol{k}_l\|_2 \leq \kappa, \quad \|\boldsymbol{x}_l\|_2 \leq \sqrt{d}, \quad D \geq d_{\min}, \end{aligned} \tag{79}$$

where $|\varphi'(x)| \leq B_\varphi$ for all x with $\varphi(x) = \mathrm{ELU}(x) + 1$.

Why is Assumption 15 reasonable? Assumption 15 is a local per-step envelope condition for norm comparison on a fixed realized forward pass at the current parameter values. For AttnRes and LLARA, the bounds are evaluated on their respective realized trajectories: AttnRes residuals from the depth-attention recurrence and LLARA residuals from the frozen backbone in Eq. (7). The bounds $\|\boldsymbol{x}_l\|_2 \leq \sqrt{d}$ and $\|\tilde{\boldsymbol{k}}_i\|_2 \leq \sqrt{d}$ follow directly from the RMSNorm definition in Eq. (18), since $\|\mathrm{RMSNorm}(\boldsymbol{v})\|_2^2 = \|\boldsymbol{v}\|_2^2/(\|\boldsymbol{v}\|_2^2/d + \delta) \leq d$ for all $\boldsymbol{v} \in \mathbb{R}^d$. The feature map $\varphi(x) = \mathrm{ELU}(x) + 1$ satisfies $|\varphi'(x)| \leq 1$, so one may take $B_\varphi = 1$ because $\varphi'(x) = 1$ for $x > 0$ and $\varphi'(x) = \exp(x) \leq 1$ for $x \leq 0$. The constraint $D \geq d_{\min}$ excludes the readout singularity and reflects the stabilizer $\varepsilon > 0$ in Eq. (13). The feature map is strictly positive but is not bounded below by one: $\varphi(x) = x + 1 > 1$ for $x > 0$, whereas $\varphi(x) = \exp(x) \in (0, 1)$ for $x < 0$, so φ takes values in $(0, \infty)$ and the value 1 is attained only at $x = 0$. At the zero initialization $\boldsymbol{A}_l = \boldsymbol{0}$ and $\boldsymbol{q} = \boldsymbol{0}$ one therefore has $\boldsymbol{k}_l = \boldsymbol{u} = \boldsymbol{1}$ exactly, hence $\|\boldsymbol{u}\|_2 = \|\boldsymbol{k}_l\|_2 = \sqrt{r}$ and $D = \boldsymbol{u}^\top \boldsymbol{z}_L = r(L+1)$ before any parameter update, so κ need only exceed $\sqrt{r}$ at the start of training. After parameter updates the entries of $\boldsymbol{u}$ and $\boldsymbol{k}_l$ may fall below one, which is compatible with the upper envelope κ but removes this initialization lower bound on D. The stabilizer therefore carries the constraint along the trajectory: one may take $d_{\min} = \varepsilon$ in general, and $d_{\min}$ on the order of r only near initialization. The envelope $\|\boldsymbol{v}_l\|_2 \leq V$ is compatible with the PreNorm residual scale in Assumption 3. On bounded inputs, a fixed forward pass yields a finite residual envelope (Xiong et al., 2020; Zhang & Sennrich, 2019). These AttnRes envelopes also supply the bounded realized set under which Assumption 14 holds. For the gradient bounds, LLARA routes $\mathcal{L}$ through the readout correction $\Delta \boldsymbol{h}$ in Eq. (76), so $\|\nabla_{\Delta \boldsymbol{h}} \mathcal{L}\|$ is the standard final-layer backprop signal used in one-step SGD analysis. AttnRes routes $\mathcal{L}$ through the final depth-attention aggregate $\boldsymbol{h}_L$, and when only $\boldsymbol{h}_L$ enters the loss, $\|\nabla_{\boldsymbol{\zeta}_L} \mathcal{L}\|_2$ coincides with $\|\nabla_{\boldsymbol{h}_L} \mathcal{L}\|_2$. Both quantities are routinely controlled in post-training by learning rate, weight decay, or clipping. Since LLARA trains only $\{\boldsymbol{A}_l\}_{l=0}^L$ and $\boldsymbol{q}$ with rank $r \ll d$, the envelopes $\|\boldsymbol{u}\|_2, \|\boldsymbol{k}_l\|_2 \leq \kappa$ follow from low-rank projection scale and a bounded parameter domain maintained at each step under learning rate and weight decay.

G.2 AttnRes Gradient Derivation

We first derive the local query gradient at a fixed block l, then extend to the full recurrence.

Softmax Jacobian. Fix l and write $s_i = s_{i \to l}$ and $\alpha_i = \alpha_{i \to l}$ for $i = 0, \ldots, l$. For the softmax map $s \mapsto \alpha(s)$ with $\alpha_i = \exp(s_i)/\sum_j \exp(s_j)$, the Jacobian entries are:

$$\frac{\partial \alpha_i}{\partial s_j} = \alpha_i(\delta_{ij} - \alpha_j), \quad 0 \leq i, j \leq l. \tag{80}$$

Gradient of the aggregate with respect to scores. Since $\boldsymbol{h}_l = \sum_{i=0}^l \alpha_i \boldsymbol{v}_i$, for each $0 \leq j \leq l$:

$$\frac{\partial \boldsymbol{h}_l}{\partial s_j} = \sum_{i=0}^{l} \boldsymbol{v}_i \frac{\partial \alpha_i}{\partial s_j} = \sum_{i=0}^{l} \alpha_i(\delta_{ij} - \alpha_j)\boldsymbol{v}_i = \alpha_j(\boldsymbol{v}_j - \boldsymbol{h}_l), \tag{81}$$

using $\sum_i \alpha_i = 1$ and $\boldsymbol{h}_l = \sum_i \alpha_i \boldsymbol{v}_i$.

Score-to-query map. Because $s_j = \boldsymbol{q}_l^\top \tilde{\boldsymbol{k}}_j$, we have $\partial s_j / \partial \boldsymbol{q}_l = \tilde{\boldsymbol{k}}_j$. Let $\boldsymbol{g}_l := \nabla_{\boldsymbol{h}_l} \mathcal{L}$ denote the upstream gradient for this local calculation. For a scalar loss, $\partial \mathcal{L}/\partial s_j = \boldsymbol{g}_l^\top \partial \boldsymbol{h}_l / \partial s_j$ is a scalar, hence:

$$\nabla_{\boldsymbol{q}_l} \mathcal{L}\big|_{\mathrm{local}} = \sum_{j=0}^{l} \frac{\partial \mathcal{L}}{\partial s_j} \frac{\partial s_j}{\partial \boldsymbol{q}_l} = \sum_{j=0}^{l} \left(\boldsymbol{g}_l^\top \partial \boldsymbol{h}_l / \partial s_j\right) \tilde{\boldsymbol{k}}_j = \sum_{j=0}^{l} \alpha_j \left(\boldsymbol{g}_l^\top (\boldsymbol{v}_j - \boldsymbol{h}_l)\right) \tilde{\boldsymbol{k}}_j, \tag{82}$$

using Eq. (81) and $\partial s_j/\partial \boldsymbol{q}_l = \tilde{\boldsymbol{k}}_j$.

Recurrence backpropagation. The local gradient in Eq. (82) differentiates only the softmax map at block l while holding $\{\boldsymbol{v}_i\}_{i=0}^{l}$ fixed. In the AttnRes recurrence, perturbing $\boldsymbol{q}_l$ changes $\boldsymbol{h}_l$, then $\boldsymbol{v}_{l+1} = \mathcal{F}_{l+1}(\boldsymbol{h}_l)$, and then all later values, keys, softmax weights, and aggregates. We therefore separate the exact local aggregate-to-query Jacobian $\boldsymbol{M}_l = \partial \boldsymbol{h}_l/\partial \boldsymbol{q}_l$ from the recurrent adjoint arriving at $\boldsymbol{h}_l$.

Hold $\{\boldsymbol{v}_i\}_{i=0}^{l}$ fixed in Eq. (74) and vary only $\boldsymbol{q}_l$. Writing $\mathrm{d}s_j = \tilde{\boldsymbol{k}}_j^\top \mathrm{d}\boldsymbol{q}_l$ and applying Eq. (81) yields:

$$\mathrm{d}\boldsymbol{h}_l = \sum_{j=0}^{l} \alpha_j(\boldsymbol{v}_j - \boldsymbol{h}_l)\,\mathrm{d}s_j = \left(\sum_{j=0}^{l} \alpha_j(\boldsymbol{v}_j - \boldsymbol{h}_l)\,\tilde{\boldsymbol{k}}_j^\top\right)\mathrm{d}\boldsymbol{q}_l. \tag{83}$$

Equivalently, with $\boldsymbol{M}_l := \partial \boldsymbol{h}_l/\partial \boldsymbol{q}_l \in \mathbb{R}^{d\times d}$:

$$\boldsymbol{M}_l = \frac{\partial \boldsymbol{h}_l}{\partial \boldsymbol{q}_l} = \sum_{j=0}^{l} \alpha_j(\boldsymbol{v}_j - \boldsymbol{h}_l)\,\tilde{\boldsymbol{k}}_j^\top. \tag{84}$$

For $m \geq l+1$, write:

$$\boldsymbol{B}_m := \frac{\partial \mathcal{F}_m}{\partial \boldsymbol{h}}\Big|_{\boldsymbol{h}_{m-1}}, \quad \boldsymbol{R}_i := \frac{\partial\,\mathrm{RMSNorm}(\boldsymbol{v}_i)}{\partial \boldsymbol{v}_i}. \tag{85}$$

Let $\boldsymbol{H}_t := \partial \boldsymbol{h}_t/\partial \boldsymbol{q}_l$ and $\boldsymbol{V}_t := \partial \boldsymbol{v}_t/\partial \boldsymbol{q}_l$. The base sensitivity is:

$$\boldsymbol{H}_l = \boldsymbol{M}_l. \tag{86}$$

For later residual values, the chain rule gives:

$$\boldsymbol{V}_m = \boldsymbol{B}_m \boldsymbol{H}_{m-1}, \quad m \geq l+1, \tag{87}$$

and $\boldsymbol{V}_i = \boldsymbol{0}$ for $i \leq l$. For a later aggregate $m > l$, both the values and the keys inside the scores move. Differentiating $\boldsymbol{h}_m = \sum_{i=0}^{m} \alpha_{i\to m}\boldsymbol{v}_i$ gives:

$$\mathrm{d}\boldsymbol{h}_m = \sum_{i=0}^{m} \alpha_{i\to m}\,\mathrm{d}\boldsymbol{v}_i + \sum_{i=0}^{m} \alpha_{i\to m}(\boldsymbol{v}_i - \boldsymbol{h}_m)\,\mathrm{d}s_{i\to m}. \tag{88}$$

Since $\boldsymbol{q}_m$ is fixed when differentiating with respect to $\boldsymbol{q}_l$ and $s_{i\to m} = \boldsymbol{q}_m^\top \mathrm{RMSNorm}(\boldsymbol{v}_i)$, the score variation is:

$$\mathrm{d}s_{i\to m} = \boldsymbol{q}_m^\top \boldsymbol{R}_i\,\mathrm{d}\boldsymbol{v}_i. \tag{89}$$

Substituting Eqs. (87) and (89) into Eq. (88) gives the exact sensitivity recursion:

$$\boldsymbol{H}_m = \sum_{i=l+1}^{m} \alpha_{i\to m}\left[\boldsymbol{I}_d + (\boldsymbol{v}_i - \boldsymbol{h}_m)\,\boldsymbol{q}_m^\top \boldsymbol{R}_i\right]\boldsymbol{V}_i, \quad \boldsymbol{V}_i = \boldsymbol{B}_i \boldsymbol{H}_{i-1}. \tag{90}$$

This recursion is the full expansion of $\partial \boldsymbol{v}_i/\partial \boldsymbol{q}_l$ and $\partial \boldsymbol{h}_m/\partial \boldsymbol{q}_l$: the term $\boldsymbol{V}_i$ propagates through the block Jacobian $\boldsymbol{B}_i$, while $(\boldsymbol{v}_i - \boldsymbol{h}_m)\boldsymbol{q}_m^\top \boldsymbol{R}_i \boldsymbol{V}_i$ is the additional path through the key-dependent softmax score at the later aggregate.

Let $\boldsymbol{\zeta}_m = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_m, \boldsymbol{h}_m)$ be the AttnRes state and $\boldsymbol{\zeta}_m = \mathcal{T}_m(\boldsymbol{\zeta}_{m-1}; \boldsymbol{q}_m)$ its transition. Let $\bar{\boldsymbol{\eta}}_l := \nabla_{\boldsymbol{\zeta}_l}\mathcal{L}$ be the total reverse-mode adjoint of the state at depth l, and let $\bar{\boldsymbol{g}}_l$ denote its component with respect to $\boldsymbol{h}_l$. Because $\boldsymbol{q}_l$ directly enters the computation only through $\boldsymbol{h}_l = \sum_{i=0}^{l} \alpha_{i\to l}\boldsymbol{v}_i$, the loss variation with respect to $\boldsymbol{q}_l$ is:

$$\mathrm{d}\mathcal{L} = \bar{\boldsymbol{g}}_l^\top \mathrm{d}\boldsymbol{h}_l = \bar{\boldsymbol{g}}_l^\top \boldsymbol{M}_l\,\mathrm{d}\boldsymbol{q}_l. \tag{91}$$

Therefore the exact AttnRes query gradient is:

$$\nabla_{\boldsymbol{q}_l}\mathcal{L} = \boldsymbol{M}_l^\top \bar{\boldsymbol{g}}_l = \sum_{j=0}^{l} \alpha_{j\to l}\big(\bar{\boldsymbol{g}}_l^\top(\boldsymbol{v}_j - \boldsymbol{h}_l)\big)\tilde{\boldsymbol{k}}_j. \tag{92}$$

The future recurrence affects Eq. (92) only through the adjoint $\bar{\boldsymbol{g}}_l$. Eq. (90) shows explicitly why replacing $\bar{\boldsymbol{g}}_l$ by independent local gradients from all later depths would double-count paths and would omit the key-dependent score variations.

Proposition 16 (AttnRes query-gradient bound)**.** *Let $\bar{\boldsymbol{g}}_l$ be the $\boldsymbol{h}_l$ component of the total AttnRes state adjoint $\bar{\boldsymbol{\eta}}_l = \nabla_{\boldsymbol{\zeta}_l}\mathcal{L}$. Then:*

$$\|\nabla_{\boldsymbol{q}_l}\mathcal{L}\|_2 = \|\boldsymbol{M}_l^\top \bar{\boldsymbol{g}}_l\|_2 \le 2\sqrt{d}\,V\,\|\bar{\boldsymbol{g}}_l\|_2. \tag{93}$$

Proof. Eq. (92) gives $\nabla_{\boldsymbol{q}_l}\mathcal{L} = \boldsymbol{M}_l^\top \bar{\boldsymbol{g}}_l$. For any vector $\boldsymbol{a}$:

$$\|\boldsymbol{M}_l^\top \boldsymbol{a}\|_2 = \left\|\sum_{j=0}^{l} \alpha_j \big(\boldsymbol{a}^\top(\boldsymbol{v}_j - \boldsymbol{h}_l)\big)\tilde{\boldsymbol{k}}_j\right\|_2 \le \sum_{j=0}^{l} \alpha_j \|\boldsymbol{a}\|_2\,\|\boldsymbol{v}_j - \boldsymbol{h}_l\|_2\,\|\tilde{\boldsymbol{k}}_j\|_2 \le 2\sqrt{d}\,V\,\|\boldsymbol{a}\|_2, \tag{94}$$

using $\sum_j \alpha_j = 1$, $\|\tilde{\boldsymbol{k}}_j\|_2 \le \sqrt{d}$ from Eq. (18), and $\|\boldsymbol{v}_j - \boldsymbol{h}_l\|_2 \le 2V$. Set $\boldsymbol{a} = \bar{\boldsymbol{g}}_l$. □

Why later local sums are not exact. Only the block-l scores $s_{j\to l} = \boldsymbol{q}_l^\top \tilde{\boldsymbol{k}}_j$ depend directly on $\boldsymbol{q}_l$. For any later depth $m > l$, the scores $s_{i\to m} = \boldsymbol{q}_m^\top \mathrm{RMSNorm}(\boldsymbol{v}_i)$ contain the different parameter $\boldsymbol{q}_m$; their dependence on $\boldsymbol{q}_l$ is only through the propagated values $\boldsymbol{v}_i(\boldsymbol{q}_l)$. Let $\boldsymbol{a}_m$ be an adjoint applied to $\boldsymbol{h}_m$ and define:

$$\boldsymbol{C}_{i,m} := \alpha_{i\to m}\left[\boldsymbol{I}_d + (\boldsymbol{v}_i - \boldsymbol{h}_m)\boldsymbol{q}_m^\top \boldsymbol{R}_i\right]. \tag{95}$$

Then Eq. (90) can be written as:

$$\boldsymbol{H}_m = \sum_{i=l+1}^{m} \boldsymbol{C}_{i,m}\boldsymbol{B}_i\boldsymbol{H}_{i-1}, \quad m > l, \tag{96}$$

and the exact contribution of this later aggregate to a variation of $\boldsymbol{q}_l$ is:

$$\boldsymbol{a}_m^\top \mathrm{d}\boldsymbol{h}_m = \boldsymbol{a}_m^\top \boldsymbol{H}_m\,\mathrm{d}\boldsymbol{q}_l = \left[\sum_{i=l+1}^{m} \boldsymbol{H}_{i-1}^\top \boldsymbol{B}_i^\top \boldsymbol{C}_{i,m}^\top \boldsymbol{a}_m\right]^\top \mathrm{d}\boldsymbol{q}_l. \tag{97}$$

Thus a later-depth contribution is mediated by the already accumulated sensitivities $\boldsymbol{H}_{i-1}$ and by the block Jacobians $\boldsymbol{B}_i$. In contrast, the independent local score-to-query gradient at depth m would be:

$$\nabla_{\boldsymbol{q}_m}\mathcal{L}_m|_{\mathrm{local}} = \sum_{i=0}^{m} \alpha_{i\to m}\big(\boldsymbol{a}_m^\top(\boldsymbol{v}_i - \boldsymbol{h}_m)\big)\tilde{\boldsymbol{k}}_i, \tag{98}$$

which differentiates with respect to $\boldsymbol{q}_m$ while holding the values fixed. Eq. (98) is therefore not a gradient with respect to $\boldsymbol{q}_l$ for $m > l$: it omits the value-propagation factors $\boldsymbol{H}_{i-1}^\top \boldsymbol{B}_i^\top$ and also omits the key-variation operator $\boldsymbol{R}_i^\top \boldsymbol{q}_m(\boldsymbol{v}_i - \boldsymbol{h}_m)^\top$ contained in $\boldsymbol{C}_{i,m}^\top$. Moreover, once reverse mode is used, all effects of $\boldsymbol{h}_l \to \boldsymbol{\zeta}_{l+1} \to \cdots \to \boldsymbol{\zeta}_L$ are already collected in the total adjoint $\bar{\boldsymbol{g}}_l$. Adding independent local terms such as Eq. (98) would count downstream paths outside the single chain-rule composition and would not reproduce the exact variation in Eq. (97).

Jacobian product structure. Equations (87)-(90) are the forward sensitivities induced by $\boldsymbol{q}_l$ through later residual values, keys, softmax weights, and aggregates. The reverse-mode statement is obtained by differentiating the same state transitions. Define the transition Jacobian:

$$\boldsymbol{J}_m := \frac{\partial \mathcal{T}_m}{\partial \boldsymbol{\zeta}_{m-1}}, \quad m = l+1, \ldots, L, \tag{99}$$

with $\boldsymbol{q}_m$ held fixed in this state derivative. For state adjoints $\bar{\boldsymbol{\eta}}_m = \nabla_{\boldsymbol{\zeta}_m}\mathcal{L}$, the chain rule gives:

$$\bar{\boldsymbol{\eta}}_{m-1} = \boldsymbol{J}_m^\top \bar{\boldsymbol{\eta}}_m, \quad m = l+1, \ldots, L, \tag{100}$$

and hence:

$$\bar{\boldsymbol{\eta}}_l = \boldsymbol{J}_{l+1}^\top \boldsymbol{J}_{l+2}^\top \cdots \boldsymbol{J}_L^\top \bar{\boldsymbol{\eta}}_L. \tag{101}$$

If $\boldsymbol{P}_l$ denotes the projection from the full state adjoint $\bar{\boldsymbol{\eta}}_l$ onto its $\boldsymbol{h}_l$ component, then $\bar{\boldsymbol{g}}_l = \boldsymbol{P}_l\bar{\boldsymbol{\eta}}_l$ and $\|\boldsymbol{P}_l\|_2 \le 1$. Combining Eq. (92) with Eq. (101) gives the exact product form:

$$\nabla_{\boldsymbol{q}_l}\mathcal{L} = \boldsymbol{M}_l^\top \boldsymbol{P}_l \boldsymbol{J}_{l+1}^\top \boldsymbol{J}_{l+2}^\top \cdots \boldsymbol{J}_L^\top \bar{\boldsymbol{\eta}}_L, \tag{102}$$

with the empty product equal to the identity when $l = L$. Taking operator norms and using Proposition 16 together with Assumptions 14 and 15 yields:

$$\|\nabla_{\boldsymbol{q}_l}\mathcal{L}\|_2 \le C_{\mathrm{loc}}\,\|\boldsymbol{P}_l\|_2 \left(\prod_{m=l+1}^{L} \|\boldsymbol{J}_m\|_2\right) \|\bar{\boldsymbol{\eta}}_L\|_2 \le \Big(\prod_{m=l+1}^{L} L_{\mathrm{AttnRes}}\Big) C_{\mathrm{loc}} G, \tag{103}$$

where $C_{\mathrm{loc}} = 2\sqrt{d}\,V$. This is the depth-recurrent factor used in Theorem 1(i).

G.3 LLARA Readout Gradient Derivation

We derive gradients for the normalized readout in Eq. (76). Throughout, $\boldsymbol{S}_L \in \mathbb{R}^{r\times d}$, $\boldsymbol{u}, \boldsymbol{k}_l, \boldsymbol{z}_L \in \mathbb{R}^r$, and $\boldsymbol{v}_l, \Delta\boldsymbol{h}, \boldsymbol{g} \in \mathbb{R}^d$ are column vectors. The readout is equivalently $\Delta\boldsymbol{h} = \boldsymbol{S}_L^\top \boldsymbol{u}/(\boldsymbol{u}^\top \boldsymbol{z}_L)$, matching the implementation contraction $\sum_j u_j (\boldsymbol{S}_L)_{j,:}$.

Quotient rule for the readout. Write $N = \boldsymbol{S}_L^\top \boldsymbol{u} \in \mathbb{R}^d$ and $D = \boldsymbol{u}^\top \boldsymbol{z}_L \in \mathbb{R}$, so $\Delta\boldsymbol{h} = N/D$. For a scalar loss $\mathcal{L}$ with $\boldsymbol{g} = \nabla_{\Delta\boldsymbol{h}}\mathcal{L}$, the partial derivatives are:

$$\frac{\partial N}{\partial \boldsymbol{u}} = \boldsymbol{S}_L^\top \in \mathbb{R}^{d\times r}, \quad \frac{\partial D}{\partial \boldsymbol{u}} = \boldsymbol{z}_L \in \mathbb{R}^r, \tag{104}$$

since $N_i = \sum_a (\boldsymbol{S}_L)_{a,i} u_a$ and $D = \sum_a u_a (\boldsymbol{z}_L)_a$. Applying the vector quotient rule yields:

$$\nabla_{\boldsymbol{u}}\mathcal{L} = \frac{1}{D}\left(\frac{\partial N}{\partial \boldsymbol{u}}\right)^\top \boldsymbol{g} - \frac{\boldsymbol{g}^\top N}{D^2}\,\boldsymbol{z}_L = \frac{1}{D}\boldsymbol{S}_L\boldsymbol{g} - \frac{\boldsymbol{g}^\top N}{D^2}\,\boldsymbol{z}_L = \frac{1}{D}\Big(\boldsymbol{S}_L\boldsymbol{g} - \big(\boldsymbol{g}^\top \Delta\boldsymbol{h}\big)\boldsymbol{z}_L\Big), \tag{105}$$

using $N/D = \Delta\boldsymbol{h}$. Since $\boldsymbol{u} = \varphi(\boldsymbol{q})$ is applied elementwise, the chain rule gives:

$$\nabla_{\boldsymbol{q}}\mathcal{L} = \operatorname{diag}\big(\varphi'(\boldsymbol{q})\big)\,\nabla_{\boldsymbol{u}}\mathcal{L} = \varphi'(\boldsymbol{q}) \odot \nabla_{\boldsymbol{u}}\mathcal{L}, \tag{106}$$

where $\varphi'(x) = 1$ if $x > 0$ and $\varphi'(x) = \exp(x)$ if $x \le 0$ for $\varphi(x) = \mathrm{ELU}(x) + 1$. The LLARA query is not depth-local because $\boldsymbol{u}$ appears in every weight $w_i = \boldsymbol{u}^\top \boldsymbol{k}_i$, but its gradient still passes through only this single readout. Indeed, using $\boldsymbol{S}_L = \sum_{i=0}^L \boldsymbol{k}_i \boldsymbol{v}_i^\top$ and $\boldsymbol{z}_L = \sum_{i=0}^L \boldsymbol{k}_i$ in Eq. (105) gives:

$$\nabla_{\boldsymbol{u}}\mathcal{L} = \frac{1}{D}\sum_{i=0}^{L} \boldsymbol{k}_i\, \boldsymbol{g}^\top(\boldsymbol{v}_i - \Delta\boldsymbol{h}). \tag{107}$$

Consequently, under Assumption 15:

$$\|\nabla_{\boldsymbol{u}}\mathcal{L}\|_2 \le \frac{1}{D}\sum_{i=0}^{L} \|\boldsymbol{k}_i\|_2\, \|\boldsymbol{g}\|_2\, \|\boldsymbol{v}_i - \Delta\boldsymbol{h}\|_2 \le \frac{2(L+1)\kappa G V}{d_{\min}}. \tag{108}$$

Here $\|\boldsymbol{v}_i - \Delta\boldsymbol{h}\|_2 \le 2V$ follows because $\Delta\boldsymbol{h}$ is a nonnegative normalized combination of the bounded residual increments (or the same combination scaled by a factor at most one when the implementation stabilizer $\varepsilon > 0$ is included). Combining this with Eq. (106) yields:

$$\|\nabla_{\boldsymbol{q}}\mathcal{L}\|_2 \le B_\varphi \|\nabla_{\boldsymbol{u}}\mathcal{L}\|_2 \le \frac{2B_\varphi(L+1)\kappa G V}{d_{\min}}. \tag{109}$$

The factor $L + 1$ reflects that the LLARA query participates in one normalized readout over all residual increments; it is not a product of depth-transition Jacobians.

Intermediate gradient with respect to a single key. Fix $l \in \{0, \dots, L\}$ and treat $\boldsymbol{k}_l$ as an independent variable in the sums defining $\boldsymbol{S}_L$ and $\boldsymbol{z}_L$:

$$\boldsymbol{S}_L = \boldsymbol{k}_l \boldsymbol{v}_l^\top + \sum_{i\ne l} \boldsymbol{k}_i \boldsymbol{v}_i^\top, \quad \boldsymbol{z}_L = \boldsymbol{k}_l + \sum_{i\ne l} \boldsymbol{k}_i. \tag{110}$$

Only the l-th summands depend on $\boldsymbol{k}_l$. Decomposing $N = (\boldsymbol{u}^\top \boldsymbol{k}_l)\boldsymbol{v}_l + \tilde{N}$ and $D = \boldsymbol{u}^\top \boldsymbol{k}_l + \tilde{D}$ with $\tilde{N}, \tilde{D}$ independent of $\boldsymbol{k}_l$, we obtain:

$$\frac{\partial N}{\partial \boldsymbol{k}_l} = \boldsymbol{v}_l \boldsymbol{u}^\top \in \mathbb{R}^{d\times r}, \quad \frac{\partial D}{\partial \boldsymbol{k}_l} = \boldsymbol{u} \in \mathbb{R}^r, \tag{111}$$

since $\partial(\boldsymbol{u}^\top \boldsymbol{k}_l)/\partial \boldsymbol{k}_l = \boldsymbol{u}$. Applying the quotient rule, the Jacobian $\partial\Delta\boldsymbol{h}/\partial\boldsymbol{k}_l \in \mathbb{R}^{d\times r}$ is:

$$\frac{\partial \Delta\boldsymbol{h}}{\partial \boldsymbol{k}_l} = \frac{1}{D}\boldsymbol{v}_l\boldsymbol{u}^\top - \frac{1}{D}\,\Delta\boldsymbol{h}\,\boldsymbol{u}^\top = \frac{1}{D}\big(\boldsymbol{v}_l - \Delta\boldsymbol{h}\big)\boldsymbol{u}^\top. \tag{112}$$

For a scalar loss, therefore:

$$\nabla_{\boldsymbol{k}_l}\mathcal{L} = \left(\frac{\partial \Delta\boldsymbol{h}}{\partial \boldsymbol{k}_l}\right)^\top \boldsymbol{g} = \frac{\boldsymbol{u}}{D}\,\boldsymbol{g}^\top(\boldsymbol{v}_l - \Delta\boldsymbol{h}), \tag{113}$$

since $\boldsymbol{g}^\top(\boldsymbol{v}_l - \Delta\boldsymbol{h})$ is a scalar.

Gradient with respect to the projection matrix. Since $\boldsymbol{k}_l = \varphi(\boldsymbol{A}_l \boldsymbol{x}_l)$ with $\boldsymbol{x}_l = \mathrm{RMSNorm}(\boldsymbol{v}_l)$, define $\boldsymbol{y}_l = \boldsymbol{A}_l \boldsymbol{x}_l \in \mathbb{R}^r$. Elementwise differentiation yields $\partial \boldsymbol{k}_l / \partial \boldsymbol{y}_l = \mathrm{diag}(\varphi'(\boldsymbol{y}_l))$. For $\boldsymbol{y}_l = \boldsymbol{A}_l \boldsymbol{x}_l$, the matrix-calculus identity $\nabla_{\boldsymbol{A}_l} \mathcal{L} = (\nabla_{\boldsymbol{y}_l} \mathcal{L})\, \boldsymbol{x}_l^\top$ gives:

$$\nabla_{\boldsymbol{A}_l} \mathcal{L} = \mathrm{diag}\big(\varphi'(\boldsymbol{y}_l)\big)\, \nabla_{\boldsymbol{k}_l} \mathcal{L}\, \boldsymbol{x}_l^\top = \Big(\varphi'(\boldsymbol{y}_l) \odot \nabla_{\boldsymbol{k}_l} \mathcal{L}\Big)\, \boldsymbol{x}_l^\top = \frac{\varphi'(\boldsymbol{y}_l) \odot \boldsymbol{u}}{D} \big(\boldsymbol{g}^\top (\boldsymbol{v}_l - \Delta \boldsymbol{h})\big)\, \boldsymbol{x}_l^\top, \tag{114}$$

where $\odot$ denotes the Hadamard product and $\varphi'(x) = 1$ for $x > 0$, $\varphi'(x) = \exp(x)$ for $x \le 0$.

RMSNorm gradient. Write $\boldsymbol{x} = \mathrm{RMSNorm}(\boldsymbol{v})$ as in Eq. (18). Differentiating $\boldsymbol{x}_i = v_i / \mathrm{rms}(\boldsymbol{v})$ yields:

$$\frac{\partial \boldsymbol{x}_i}{\partial v_j} = \frac{\delta_{ij}}{\mathrm{rms}(\boldsymbol{v})} - \frac{v_i}{\mathrm{rms}(\boldsymbol{v})^2} \frac{\partial \mathrm{rms}(\boldsymbol{v})}{\partial v_j}, \quad \frac{\partial \mathrm{rms}(\boldsymbol{v})}{\partial v_j} = \frac{v_j}{d\, \mathrm{rms}(\boldsymbol{v})}, \tag{115}$$

and therefore:

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{v}} = \frac{1}{\mathrm{rms}(\boldsymbol{v})} \boldsymbol{I}_d - \frac{\boldsymbol{v}\boldsymbol{v}^\top}{d\, \mathrm{rms}(\boldsymbol{v})^3} = \frac{1}{\mathrm{rms}(\boldsymbol{v})} \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}\boldsymbol{v}^\top}{d\, \mathrm{rms}(\boldsymbol{v})^2}\right). \tag{116}$$

When the backbone is frozen, $\boldsymbol{v}_l$ is constant with respect to $\boldsymbol{A}_l$ and the RMSNorm chain does not contribute to $\nabla_{\boldsymbol{A}_l} \mathcal{L}$.

G.4 Proof of Theorem 1

Proof of item (i) (AttnRes). By Eq. (92) and Proposition 16:

$$\|\nabla_{\boldsymbol{q}_l} \mathcal{L}\|_2 \le C_{\mathrm{loc}}\, \|\bar{\boldsymbol{g}}_l\|_2, \quad C_{\mathrm{loc}} := 2\sqrt{d}\, V. \tag{117}$$

The state adjoint $\bar{\boldsymbol{\eta}}_l = \nabla_{\boldsymbol{\zeta}_l} \mathcal{L}$ backpropagates through the AttnRes state transitions $\mathcal{T}_{l+1}, \ldots, \mathcal{T}_L$. Let

$$\boldsymbol{J}_m := \frac{\partial \mathcal{T}_m}{\partial \boldsymbol{\zeta}_{m-1}}, \quad m = l+1, \ldots, L. \tag{118}$$

Reverse-mode differentiation through $\boldsymbol{\zeta}_m = \mathcal{T}_m(\boldsymbol{\zeta}_{m-1}; \boldsymbol{q}_m)$ gives the adjoint recurrence:

$$\bar{\boldsymbol{\eta}}_{m-1} = \boldsymbol{J}_m^\top \bar{\boldsymbol{\eta}}_m, \quad m = l+1, \ldots, L. \tag{119}$$

Hence, by the operator-norm bound in Assumption 14:

$$\|\bar{\boldsymbol{\eta}}_{m-1}\|_2 \le \|\boldsymbol{J}_m\|_2\, \|\bar{\boldsymbol{\eta}}_m\|_2 \le L_{\mathrm{AttnRes}}\, \|\bar{\boldsymbol{\eta}}_m\|_2. \tag{120}$$

Iterating from $m = L$ down to $m = l+1$ and using $\|\bar{\boldsymbol{\eta}}_L\|_2 \le G$ from Assumption 15 yields:

$$\|\bar{\boldsymbol{g}}_l\|_2 \le \|\bar{\boldsymbol{\eta}}_l\|_2 \le \Big(\prod_{m=l+1}^{L} L_{\mathrm{AttnRes}}\Big)\, G, \tag{121}$$

where the empty product equals 1 when $l = L$. Combining the two displays gives:

$$\|\nabla_{\boldsymbol{q}_l} \mathcal{L}\|_2 \le \Big(\prod_{m=l+1}^{L} L_{\mathrm{AttnRes}}\Big)\, C_{\mathrm{loc}}\, G. \tag{122}$$

Zero initialization (AttnRes). If $\boldsymbol{q}_l = \boldsymbol{0}$, then $s_{i \to l} = 0$ and $\alpha_{i \to l} = 1/(l+1)$. Unless $\bar{\boldsymbol{g}}_l^\top (\boldsymbol{v}_i - \boldsymbol{h}_l) = 0$ for all i, the local contribution in Eq. (82) can be non-zero even at initialization, so AttnRes does not start from an inactive depth-aggregator map.

Proof of item (ii) (LLARA). With a frozen backbone, $\{\boldsymbol{v}_l\}$ is independent of $\{\boldsymbol{A}_l, \boldsymbol{q}\}$. Since $\boldsymbol{k}_i = \varphi(\boldsymbol{A}_i \boldsymbol{x}_i)$, the projection parameters are local in the key map: $\partial \boldsymbol{k}_i / \partial \boldsymbol{A}_l = \boldsymbol{0}$ for $i \ne l$. The readout $\Delta \boldsymbol{h} = (\sum_i (\boldsymbol{u}^\top \boldsymbol{k}_i) \boldsymbol{v}_i) / (\sum_i \boldsymbol{u}^\top \boldsymbol{k}_i)$ still couples all depths through its normalization, so the decoupling claim is only about the route from $\boldsymbol{A}_l$ to its own key $\boldsymbol{k}_l$, not about $\partial \Delta \boldsymbol{h} / \partial \boldsymbol{A}_l$ being independent of other depths. Architecture parameters are reached only through the single readout in Eq. (76), so no gradient path chains through $L - l$ block Jacobians.

Eq. (109) gives the LLARA-query bound:

$$\|\nabla_{\boldsymbol{q}}\mathcal{L}\|_2 \le \frac{2B_\varphi(L+1)\kappa GV}{d_{\min}}. \tag{123}$$

The factor $L+1$ appears because $\boldsymbol{q}$ determines the single LLARA query feature $\boldsymbol{u}$ used against every key $\boldsymbol{k}_i$; it is an output-level sum rather than a composition of depth-transition Jacobians.

To bound the trainable projection $\boldsymbol{A}_l$, first use the intermediate key gradient in Eq. (113) and Assumption 15:

$$\|\nabla_{\boldsymbol{k}_l}\mathcal{L}\|_2 \le \frac{\|\boldsymbol{u}\|_2}{D}\,|\boldsymbol{g}^\top(\boldsymbol{v}_l-\Delta\boldsymbol{h})| \le \frac{\kappa G}{d_{\min}}\,\|\boldsymbol{v}_l-\Delta\boldsymbol{h}\|_2 \le \frac{2\kappa GV}{d_{\min}}. \tag{124}$$

Here $\|\Delta\boldsymbol{h}\|_2 \le V$ because $\Delta\boldsymbol{h}$ is a nonnegative normalized combination of $\{\boldsymbol{v}_i\}$ when $\varepsilon=0$, and with $\varepsilon>0$ it is the same combination scaled by $\sum_i w_i/(\sum_i w_i+\varepsilon)\le 1$; hence $\|\boldsymbol{v}_l-\Delta\boldsymbol{h}\|_2 \le 2V$. Eq. (114) and $\|\boldsymbol{x}_l\|_2 \le \sqrt{d}$ give:

$$\|\nabla_{\boldsymbol{A}_l}\mathcal{L}\|_F \le \frac{B_\varphi\,\kappa G}{d_{\min}}\,\|\boldsymbol{v}_l-\Delta\boldsymbol{h}\|_2\,\|\boldsymbol{x}_l\|_2 \le \frac{2B_\varphi\,\kappa GV\sqrt{d}}{d_{\min}}. \tag{125}$$

The bounds depend on $G, V, \kappa, d_{\min}, B_\varphi$ but not on recurrent Jacobian products along depth.

Zero initialization (LLARA). Zero initialization gives $\boldsymbol{A}_l\boldsymbol{x}_l = \boldsymbol{0}$ and $\boldsymbol{q} = \boldsymbol{0}$, so $\boldsymbol{k}_l = \boldsymbol{u} = \varphi(\boldsymbol{0}) = \boldsymbol{1}$ because $\mathrm{ELU}(0)=0$. With $\varepsilon>0$ in Eq. (13):

$$\Delta\boldsymbol{h} = \frac{r\sum_{i=0}^{L}\boldsymbol{v}_i}{r(L+1)+\varepsilon}, \tag{126}$$

a scaled depth-uniform average of the frozen trajectory that carries no learned selectivity. When $\varepsilon=0$, Eq. (107) gives $\nabla_{\boldsymbol{u}}\mathcal{L}=\boldsymbol{0}$ and hence $\nabla_{\boldsymbol{q}}\mathcal{L}=\boldsymbol{0}$, whereas the projection gradients $\{\nabla_{\boldsymbol{A}_l}\mathcal{L}\}$ can still be non-zero through the per-depth terms $\boldsymbol{v}_l-\Delta\boldsymbol{h}$. Architectural gradients at this point are given by Eqs. (106)-(114) and, by item (ii), contain no recurrent Jacobian products along depth.

H Why Joint Training with LoRA

The limitation of a LLARA-only adapter follows from the backbone-preserving interface in §3.2. With all pretrained weights frozen and no LoRA update inside the blocks, the residual values in Eq. (7) remain exactly the pretrained trajectory. LLARA can then reweight and recombine features that the frozen model already computed, which can mitigate the depth-wise dilution induced by uniform residual accumulation. It cannot, however, introduce new within-block computations. This gap is especially consequential for instruction following, a skill that is tightly coupled to contextual attention. Later tokens must retune their query-key affinities so as to attend to the relevant instruction and context tokens, rather than to pretrained co-occurrence patterns. Architecture-level residual recomposition cannot rewrite those pairwise maps, because they are produced inside the frozen attention heads. Domain-specific SFT and sparse-reward RL therefore typically require the local maps $\mathcal{F}_l$ themselves to change, and in particular a more precise update of attention-head parameters. Otherwise the depth readout is aggregating residual increments that were never adapted to the target task. The depth pathway can nevertheless contribute to instruction following. Training it can still strengthen contextual attention, by learning which residual increments the language-modeling head should emphasize once those within-block attention maps begin to change.

Figure 6 tests this limitation on the same Qwen2.5-7B mathematical SFT protocol used in Figure 3. The jointly trained adapter (LoRA on q/v projections plus LLARA) is compared with a LLARA-only adapter that freezes every within-block map $\mathcal{F}_l$. Both runs share the MetaMathQA split, depth rank, optimizer, and schedule. The only change is that the LLARA-only run sets the LoRA rank to zero. The LLARA-only loss decreases over training, indicating that reweighting the frozen residual stream can still reduce the supervised objective. It nevertheless plateaus at a higher floor: the minimum logged loss is 0.1737 for LLARA-only versus 0.0916 for LoRA+LLARA. This gap is the empirical counterpart of the argument above. Because the attention maps receive no local update, the depth readout can only recombine residual increments never adapted to the instruction-following task.



Figure 6: Training loss on Qwen2.5-7B mathematical SFT. LoRA+LLARA versus a LLARA-only adapter under an otherwise identical recipe. Curves are exponential-moving-average smoothed.

Joint optimization is the counterpart of this argument. Because the LLARA keys $\boldsymbol{k}_l$ and values $\boldsymbol{v}_l$ are functions of the realized hidden trajectory, any LoRA-induced change to $\mathcal{F}_l$ changes the residual stream that the depth attention observes. Training the two adapters together therefore lets task-adapted residual features and the depth readout co-evolve: LoRA supplies more informative increments, and LLARA learns which depths the language-modeling head should emphasize. Freezing one side of this coupling would either leave the residual features unadapted or freeze the depth weights against a moving residual stream.

Full-parameter fine-tuning could also change $\{\mathcal{F}_l\}$, but it is incompatible with the parameter-efficient post-training setting that motivates LLARA. For a square projection of width d, Eq. (68) trains a $2r_{\mathrm{LoRA}}/d$ fraction of the dense parameters, with the same reduction in stored gradients and optimizer states, while leaving the frozen backbone shareable across task adapters. Joint training of LLARA and LoRA therefore supplies the missing within-block adaptation without the memory, storage, and deployment cost of updating a full LLM.

I Datasets and Experimental Settings

I.1 Configuration

Unless otherwise noted, experiments run on a Linux server with an Intel Xeon Gold 6240 CPU (2.60 GHz), 256 GB RAM, and a single NVIDIA A100-SXM4 40GB GPU. Fitting the main SFT and GRPO recipes on one 40GB device is the intended operating point for LLARA, which is designed for parameter-efficient post-training rather than full-model fine-tuning. PPO is the exception among the main recipes and uses two such GPUs. The larger Qwen2.5 models in the scaling study (Figure 2) are trained on a server with an Intel Xeon Platinum 8580 CPU and NVIDIA H200 NVL GPUs. Taking the math instruction tuning task as an example, our implementation is based on PyTorch 2.2.1, with CUDA 12.1 and cuDNN 8.9, running on Python 3.10.20. We build upon the HuggingFace ecosystem, including Transformers 4.57.6, Accelerate 1.13.0, and a customized PEFT fork 0.19.1 for LLARA adapter training. Model loading and tokenization rely on HuggingFace Hub 0.36.2, Datasets 4.8.4, and Tokenizers 0.22.2. General- and code-domain SFT use the same library versions. RL experiments use PyTorch 2.4.1.

I.2 Datasets and Evaluation Protocols

We train one adapter per SFT domain, so general, mathematical, and code supervision never share parameters. Reinforcement learning uses a single verifiable-reward corpus for both PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024): the GSM8K training split (Cobbe et al., 2021), with the gold final answer retained for reward computation. Table 3 lists the training corpora and Table 4

Table 3: Training corpora. Each SFT domain uses a dedicated adapter.

Stage	Domain	Corpus	Reference	Split
SFT	General	Tülu 3 SFT mixture[3]	Lambert et al. (2024)	first 100K
	Math	MetaMathQA[4]	Yu et al. (2024)	first 100K
	Code	Magicoder Evol-Instruct[5]	Wei et al. (2024)	full train
RL	Math	GSM8K	Cobbe et al. (2021)	full train

Table 4: Evaluation protocols. Acc.: accuracy. EM: exact match.

Stage	Domain	Benchmark	Reference	n-shot	Metric
SFT	General	MMLU	Hendrycks et al. (2021a)	0	Acc.
		MMLU-Pro	Wang et al. (2024)	5	EM
		TriviaQA	Joshi et al. (2017)	0	EM
SFT	Math	GSM8K	Cobbe et al. (2021)	0†	Answer acc.
		MATH	Hendrycks et al. (2021b)	0†	Answer acc.
SFT	Code	HumanEval	Chen et al. (2021)	0	pass@1
		MBPP	Austin et al. (2021)	3	pass@1
RL	Math	GSM8K	Cobbe et al. (2021)	5	Flexible EM
		MATH	Hendrycks et al. (2021b)	4	Math verify

†No in-context exemplars. The prompt ends with a chain-of-thought cue.

the evaluation protocols. Except for mathematical SFT, which uses a custom grader, all evaluations follow the task definitions of the Language Model Evaluation Harness[2].

General knowledge. The general-domain adapter is trained on the first 100K examples of the Tülu 3 SFT mixture (Lambert et al., 2024), a multi-source instruction-following corpus. From each multi-turn conversation we keep only the first user-assistant exchange and wrap it in the Alpaca instruction template[6], reducing heterogeneous sources to a common single-turn format. MMLU (Hendrycks et al., 2021a) is a 57-subject multiple-choice knowledge exam. We score 0-shot accuracy with the harness prompt, averaged over subjects. MMLU-Pro (Wang et al., 2024) is a harder multiple-choice suite with more options. We score 5-shot exact-match accuracy after the harness answer extractor, averaged over subjects, generating at most $2{,}048$ new tokens. TriviaQA (Joshi et al., 2017) is an open-domain trivia benchmark. We use the `rc.nocontext` validation split with 0-shot greedy generation until a stop delimiter and report exact match against answer aliases.

Mathematical reasoning. Mathematical SFT uses the `query` and `response` fields of MetaMathQA (Yu et al., 2024), a corpus of rewritten mathematical questions, truncated to the first 100K training examples. The corresponding evaluators for GSM8K (Cobbe et al., 2021) grade-school word problems and MATH (Hendrycks et al., 2021b) competition problems are task-specific. Each problem is wrapped in an Alpaca-style prompt ending with "Let's think step by step," decoded greedily with $1{,}024$ new tokens on GSM8K and $2{,}048$ on MATH, and graded by answer normalization with numerical and symbolic equivalence checks. RL follows the harness definitions so that PPO and GRPO are scored identically. GSM8K is 5-shot and reports flexible exact match. MATH uses the Minerva-style 4-shot protocol (Lewkowycz et al., 2022) with mathematical answer verification. Because the SFT and RL prompts and graders differ, we never compare their absolute scores.

Code generation. Code SFT uses the `instruction` and `response` fields of Magicoder Evol-Instruct (Wei et al., 2024), a set of evolved coding instructions. HumanEval (Chen et al., 2021)

[2] `https://github.com/EleutherAI/lm-evaluation-harness`
[3] `https://huggingface.co/datasets/allenai/tulu-3-sft-mixture`
[4] `https://huggingface.co/datasets/meta-math/MetaMathQA`
[5] `https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K`
[6] `https://github.com/tatsu-lab/stanford_alpaca`

measures functional correctness of programs synthesized from docstrings and is evaluated 0-shot. MBPP (Austin et al., 2021) consists of short Python programming problems and is evaluated with three demonstrations. Both use greedy decoding and pass@1. HumanEval generates at most 1,024 new tokens; MBPP uses the harness budget of 256 new tokens with a `[DONE]` stop sequence. Generated programs are executed only in an isolated environment.

I.3 Hyperparameter Configuration

Unless otherwise specified, all runs use bfloat16 computation, TF32 matrix multiplies, and gradient checkpointing. LoRA is applied to `q_proj` and `v_proj` throughout. We select the LLARA depth-attention rank r from $\{4, 8, 16, 32, 64\}$. The readout stabilizer is $\varepsilon = 10^{-6}$.

Supervised fine-tuning. All SFT runs use AdamW with cosine decay, a warmup ratio of 0.02, and zero weight decay. The per-device batch size is one and gradients are accumulated for 64 steps, yielding an effective batch size of 64. We train for at most three epochs and report the saved checkpoint with the best downstream score. General-domain and code SFT share the same adapter and optimization settings: a training maximum sequence length of 2,048, learning rate 2×10^{-4}, LoRA rank $r_{\mathrm{LoRA}} = 16$, scaling $\alpha = 32$, and dropout 0.05. The 2,048-token cutoff is a training restriction for the longer packed sequences in Tülu 3 and Magicoder Evol-Instruct, not the backbone context window, and evaluation does not inherit it. Mathematical SFT uses a maximum sequence length of 512 tokens, a higher learning rate of 2×10^{-3}, and a smaller LoRA configuration ($r_{\mathrm{LoRA}} = 8$, $\alpha = 16$), with the same dropout. The 512-token training cutoff is a property of MetaMathQA rather than of the backbone: the rewritten questions and solutions are short, so packed prompt-response sequences fit within this budget. At evaluation we instead decode greedily with 1,024 new tokens on GSM8K and 2,048 on MATH, truncating prompts against the backbone context window.

Reinforcement learning. RL experiments fine-tune Qwen2.5-3B for one epoch on GSM8K with $r_{\mathrm{LoRA}} = 16$ and $\alpha = 32$. We cap training rollouts at 512 new tokens because GSM8K grade-school solutions are short. This is a training budget rather than the backbone context window, and evaluation follows the harness protocol above. PPO uses learning rate 1×10^{-5}, rollout batch size 16, minibatch size 4, a single sample per prompt, decoding temperature 0.7, and KL coefficient 0.05. It performs four optimization epochs per rollout batch, caps the prompt length at 512, and trains a value head in addition to the adapter. The per-token reward combines the terminal verifiable correctness signal with a KL penalty to the frozen base policy. GRPO uses learning rate 1×10^{-6}, batch size 8 with gradient accumulation of 4, eight samples per prompt, decoding temperature 0.9, and KL coefficient 0.04. The schedule is cosine decay with a warmup ratio of 0.03. The reward is a binary correctness signal plus a 0.2 format bonus for a parseable final answer. The frozen reference policy is obtained by disabling the adapter, avoiding a separate reference-model copy.

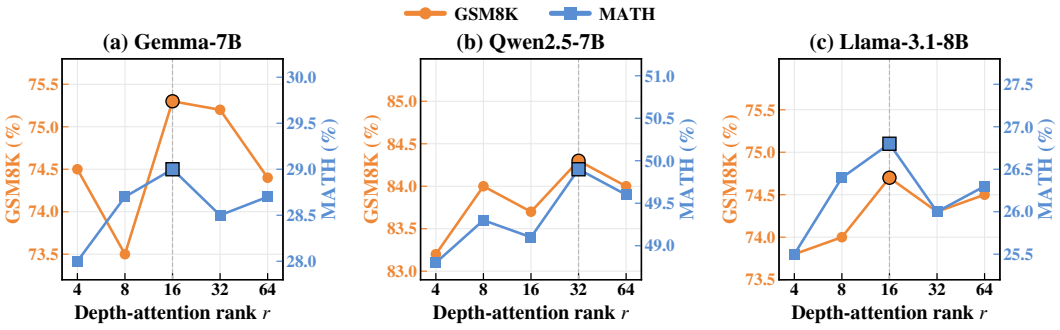


Figure 7: Sensitivity of mathematical SFT to the LLARA depth-attention rank r under a fixed LoRA configuration. **(a)** Gemma-7B. **(b)** Qwen2.5-7B. **(c)** Llama-3.1-8B. Circles (left axis) report GSM8K accuracy and squares (right axis) report MATH accuracy.

J More Information on Experiments

J.1 Sensitivity to Depth-Attention Rank

Figure 7 sweeps the LLARA depth-attention rank over $r \in \{4, 8, 16, 32, 64\}$ on mathematical SFT for Gemma-7B, Qwen2.5-7B, and Llama-3.1-8B. Both GSM8K and MATH remain in a narrow band across this range: the largest GSM8K change is 1.8 points on Gemma-7B, and the largest MATH change is 1.3 points on Llama-3.1-8B. Apart from the most compact setting $r = 4$, which is weaker on every backbone and most clearly on MATH, both benchmarks stay close to their peaks, and raising r from 16 to 64 adds no further gain. LLARA is therefore robust to the depth-attention rank in this range. That a small r already saturates the sweep supports the compact linear-attention residual pathway in §3.3: low-rank key-value trajectories and fixed-size online accumulators suffice as a parameter-efficient surrogate for depth-wise softmax mixing, without instantiating full-depth AttnRes caches or a wide depth kernel.

Table 5: Trainable parameters and systems cost of mathematical SFT on Qwen2.5-7B. Memory is the peak allocated GPU memory. Time is the one-epoch training time on a single 40GB A100.

Method	Trainable params	Trainable ratio	Memory (GB)	Time (h)
LoRA ($r = 0$)	2.52M	0.033%	15.22	5.6
LLARA ($r = 4$)	2.94M	0.039%	15.87	6.7
LLARA ($r = 8$)	3.35M	0.044%	15.94	6.4
LLARA ($r = 16$)	4.19M	0.055%	16.06	6.7
LLARA ($r = 32$)	5.85M	0.077%	16.31	7.0
LLARA ($r = 64$)	9.18M	0.120%	16.80	6.4

Table 6: Trainable parameters and systems cost of mathematical SFT on Gemma-7B.

Method	Trainable params	Trainable ratio	Memory (GB)	Time (h)
LoRA ($r = 0$)	3.21M	0.038%	17.52	6.3
LLARA ($r = 4$)	3.57M	0.042%	18.08	7.0
LLARA ($r = 8$)	3.92M	0.046%	18.14	6.8
LLARA ($r = 16$)	4.64M	0.054%	18.24	6.9
LLARA ($r = 32$)	6.06M	0.071%	18.45	6.4
LLARA ($r = 64$)	8.91M	0.104%	18.88	6.9

Table 7: Trainable parameters and systems cost of mathematical SFT on Llama-3.1-8B.

Method	Trainable params	Trainable ratio	Memory (GB)	Time (h)
LoRA ($r = 0$)	3.41M	0.042%	15.89	7.3
LLARA ($r = 4$)	3.95M	0.049%	16.73	8.3
LLARA ($r = 8$)	4.49M	0.056%	16.81	8.2
LLARA ($r = 16$)	5.57M	0.069%	16.95	8.0
LLARA ($r = 32$)	7.73M	0.096%	17.24	8.4
LLARA ($r = 64$)	12.06M	0.150%	18.07	8.8

J.2 Computational Efficiency

We profile computational cost on Qwen2.5-7B mathematical SFT under the default recipe of Appendix I: LoRA rank $r_{\mathrm{LoRA}} = 8$ on `q_proj` and `v_proj`, sequence length 512, microbatch size 1, gradient accumulation 64, bfloat16, and activation checkpointing, on a single 40GB A100. LLARA is trained jointly with this matched LoRA adapter while the depth-attention rank is swept over $r \in \{4, 8, 16, 32, 64\}$. Peak allocated GPU memory is recorded after warmup. Training time is the one-epoch runtime on a single 40GB A100, excluding dataset loading. Table 5 and Figure 8 show that the depth pathway is a small add-on to the matched LoRA baseline. LLARA trainable

parameter counts include that LoRA adapter. At the default $r = 16$, the joint run trains 4.19M parameters (0.055% of Qwen2.5-7B), adding 1.66M beyond LoRA, and peaks at 16.06GB versus 15.22GB. Even $r = 64$ remains within 9.18M trainable parameters (0.120%) and 16.80GB, so every setting in the sweep sits well below a 24GB consumer GPU budget. One-epoch training time is 5.6h for LoRA and from 6.4h to 7.0h for LLARA. The extra cost does not grow with r in this range, consistent with the $O(rd)$ online depth state in §3.4.

We repeat the same mathematical SFT recipe, LoRA configuration, rank sweep, and single-A100 profiling protocol on Gemma-7B and Llama-3.1-8B. Table 6 and Figure 9 report Gemma-7B. At the default $r = 16$, the joint run trains 4.64M parameters (0.054% of Gemma-7B), adding 1.43M beyond LoRA, and peaks at 18.24GB versus 17.52GB. Even $r = 64$ remains within 8.91M trainable parameters (0.104%) and 18.88GB. One-epoch training time is 6.3h for LoRA and from 6.4h to 7.0h for LLARA. Table 7 and Figure 10 report Llama-3.1-8B. At the default $r = 16$, the joint run trains 5.57M parameters (0.069% of Llama-3.1-8B), adding 2.16M beyond LoRA, and peaks at 16.95GB versus 15.89GB. Even $r = 64$ remains within 12.06M trainable parameters (0.150%) and 18.07GB, so every setting on both backbones sits well below a 24GB consumer GPU budget. One-epoch training time is 7.3h for LoRA and from 8.0h to 8.8h for LLARA.

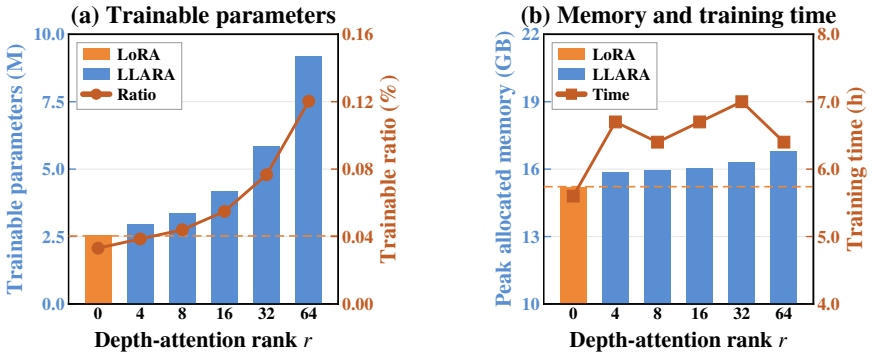


Figure 8: Computational efficiency of mathematical SFT on Qwen2.5-7B. **(a)** Trainable parameter count and its fraction of the full model. **(b)** Peak allocated GPU memory and one-epoch training time. Orange is the matched LoRA baseline plotted at $r = 0$, and blue is LLARA trained jointly with the same LoRA adapter. The dashed orange line marks the LoRA baseline.

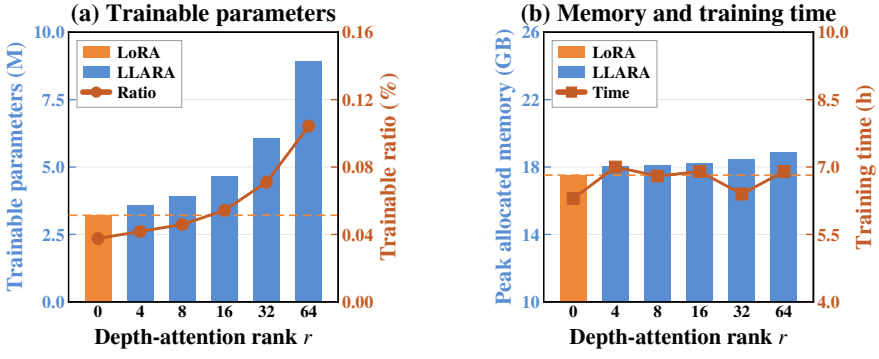


Figure 9: Computational efficiency of mathematical SFT on Gemma-7B.

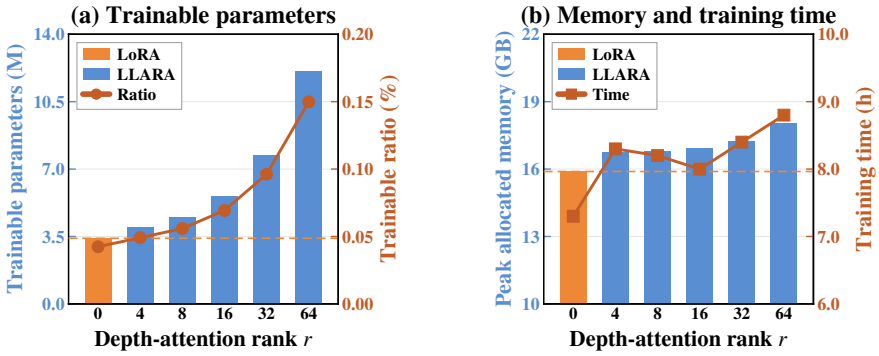


Figure 10: Computational efficiency of mathematical SFT on Llama-3.1-8B.

J.3 Extended Depth-Weight Visualizations

The main-text heatmap uses a fixed 28-token window so that three domains can be compared at a common width. This appendix reports three longer *correct* evaluation traces, one per SFT domain, under the protocols of Appendix I.2. Each record below is an inference case: instruction, test instance, greedy completion, and grader decision. These traces are not the cropped windows of Figure 4. Depth weights are logged as in §4.4: teacher-forced keys $\{\boldsymbol{k}_l\}$ over prompt and completion, $w_l = \varphi(\boldsymbol{q})^\top \boldsymbol{k}_l$, and $a_l = w_l/(\sum_{j=0}^{L} w_j + \varepsilon)$. Heatmaps show every generated token in even-length strips. A whitespace-only BPE piece is marked `sp`.

General knowledge. MMLU-Pro chemistry, index 18, 5-shot greedy generation. Gold option I, exact match.

Case A General SFT / Qwen2.5-7B

MMLU-Pro / chemistry · index 18 · Qwen2.5-7B · 5-shot greedy · exact match

Instruction

The following are multiple choice questions (with answers) about chemistry. Think step by step and then finish your answer with "the answer is (X)" where X is the correct letter choice.

Demonstration 1

Question. Predict the number of lines in the EPR spectrum of a solution of ^{13}C-labelled methyl radical ($^{13}CH_3\cdot$), assuming the lines do not overlap.

Options. A. 10 B. 8 C. 4 D. 20 E. 12
F. 3 G. 16 H. 5 I. 24 J. 6

Answer. Let's think step by step. The electron paramagnetic resonance spectrum will be split by two forms of interactions. The first is the hyperfine interaction with ^{13}C (nuclear spin $I = \frac{1}{2}$) which will split the spectrum into 2 lines. This will be further split into 4 lines by the interaction with three equivalent ^{1}H nuclei. The total number of lines is therefore $2 \cdot 4 = 8$. The answer is (B).

Demonstration 2

Question. Which of the following lists the hydrides of group-14 elements in order of thermal stability, from lowest to highest?

Options. A. $GeH_4 < SnH_4 < PbH_4 < SiH_4 < CH_4$
B. $SiH_4 < GeH_4 < SnH_4 < PbH_4 < CH_4$
C. $PbH_4 < CH_4 < SnH_4 < GeH_4 < SiH_4$
D. $PbH_4 < SnH_4 < CH_4 < GeH_4 < SiH_4$
E. $SnH_4 < GeH_4 < SiH_4 < PbH_4 < CH_4$
F. $CH_4 < GeH_4 < SnH_4 < PbH_4 < SiH_4$
G. $SiH_4 < SnH_4 < PbH_4 < GeH_4 < CH_4$
H. $CH_4 < SiH_4 < GeH_4 < SnH_4 < PbH_4$

I. $CH_4 < PbH_4 < GeH_4 < SnH_4 < SiH_4$
J. $PbH_4 < SnH_4 < GeH_4 < SiH_4 < CH_4$

Answer. Let's think step by step. The thermal stability of group-14 hydrides decreases as we move from the top of group 14 to the bottom. The order of elements in the group from top to bottom is C, Si, Ge, Sn, Pb. Therefore in order of increasing thermal stability we have PbH_4, SnH_4, GeH_4, SiH_4, and CH_4, or answer (J). The answer is (J).

Demonstration 3

Question. Which of the following is considered an acid anhydride?

Options. A. H_2SO_3 B. NaCl C. CH_4 D. HNO_3 E. SO_2
F. $Al(NO_3)_3$ G. $CaCO_3$ H. H_2O I. HCl J. N/A

Answer. Let's think step by step. An acid anhydride is a compound that is derived by removing water from an acid. The chemical formula for water is H_2O, which means that we need to determine which of these options, when combined with H_2O, forms an acid. SO_2, or sulfur dioxide, when combined with H_2O, makes H_2SO_4, or sulfuric acid. The answer is (E).

Demonstration 4

Question. A new compound is synthesized and found to be a monoprotic acid with a molar mass of 248 g/mol. When 0.0050 mol of this acid are dissolved in 0.500 L of water, the pH is measured as 3.89. What is the pK_a of this acid?

Options. A. 5.78 B. 4.78 C. 4.56 D. 6.89 E. 7.78
F. 3.89 G. 1.23 H. 2.89 I. 2.33 J. 5.33

Answer. Let's think step by step. Recall that $[A] = [H^+]$. Here, this is equal to $10^{-3.89}$. Then we have

$$K_a = \frac{[H^+][A^-]}{[HA]} = \frac{10^{-3.89} \cdot 10^{-3.89}}{10^{-2}}.$$

The resulting exponent is $-3.89 + (-3.89) - (-2) = 5.78$, therefore $K_a = 10^{-5.78}$. The pK_a is the negative log of K_a, which is equal to 5.78. The answer is (A).

Demonstration 5

Question. A solution contains 2.00 mole of acetic acid, CH_3COOH, and 1.00 mole of calcium acetate, $Ca(CH_3COO)_2$. The solution is able to resist the addition of a small amount of strong acid or strong base with only minor changes in the pH of the solution. Larger quantities of strong acid or strong base can cause a significant change in pH. How many moles of nitric acid, HNO_3, may be added before the pH begins to change significantly?

Options. A. 0.250 mole B. 0.500 mole C. 3.00 mole D. 1.00 mole E. 3.50 mole
F. 1.50 mole G. 2.50 mole H. 4.00 mole I. 0.750 mole J. 2.00 mole

Answer. Let's think step by step. We would like to compute the buffer capacity of this solution. First we write the equation for the ionization of the weak acid, in this case of acetic acid: $CH_3COOH(aq) + H_2O \rightarrow H_3O^+ + CH_3COO^-$. The conjugate base is therefore the acetate ion. The added strong acid, nitric acid, will react with the conjugate base. Therefore the maximum amount of acid that can be added will be equal to the amount of acetate ion, or 2 moles. The answer is (J).

Test item

Question. An electron confined to a metallic nanoparticle is modelled as a particle in a one-dimensional box of length L. If the electron is in the state $n = 1$, calculate the probability of finding it in $0 \le x \le \frac{1}{2}L$.

Options. A. $\frac{1}{4}$ B. $\frac{7}{8}$ C. $\frac{1}{8}$ D. $\frac{3}{4}$ E. $\frac{2}{3}$
F. $\frac{1}{3}$ G. $\frac{3}{8}$ H. $\frac{5}{8}$ I. $\frac{1}{2}$ J. $\frac{5}{6}$

Answer: Let's think step by step.

Completion

The probability is the integral of ψ^2 over that region, with $\psi(x) = \sqrt{2/L}\,\sin(\pi x/L)$ for $n = 1$:

$$P = \int_0^{\frac{1}{2}L} \frac{2}{L} \sin^2\!\left(\frac{\pi x}{L}\right) dx = \int_0^{\frac{1}{2}L} \frac{1}{L}\left(1 - \cos\frac{2\pi x}{L}\right) dx.$$

Evaluating the antiderivative gives $P = \frac{1}{2} - \frac{1}{2\pi}\sin\pi = \frac{1}{2}$. The answer is (I).

Grade

gold I · extracted I · exact match = 1

Figure 11 stays in the late-decoder band of Figure 4(a) over a much longer trace. Argmax mass concentrates on $\ell \in \{19, 21, 22, 23, 24, 25\}$, including the integral fragments and the closing letter `(I)`. Early blocks rarely win a column. The late-depth preference is therefore a property of the whole completion, not of the final option token.

Mathematical reasoning. GSM8K test index 331, 0-shot Alpaca prompt with a chain-of-thought cue. Gold answer 8400. We first record Qwen2.5-7B, then repeat the same item on the Gemma-7B and Llama-3.1-8B mathematical SFT adapters of Table 1.

Case B Mathematical SFT / Qwen2.5-7B

GSM8K · index 331 · Qwen2.5-7B · 0-shot greedy · numerical accuracy

Prompt

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:
Nick is choosing between two jobs. Job A pays $15 an hour for 2000 hours a year, and is in a state with a 20% total tax rate. Job B pays $42,000 a year and is in a state that charges $6,000 in property tax and a 10% tax rate on net income after property tax. How much more money will Nick make at the job with a higher net pay rate, compared to the other job?

Response: Let's think step by step.

Completion

For Job A, Nick will earn $15 an hour × 2000 hours = $30,000 a year. The total tax rate is 20%, so Nick will pay $30,000 × 0.20 = $6,000 in taxes. Therefore, Nick's net pay for Job A is $30,000 - $6,000 = $24,000. For Job B, Nick will earn $42,000 a year. The property tax is $6,000, so Nick's net income after property tax is $42,000 - $6,000 = $36,000. The tax rate on net income is 10%, so Nick will pay $36,000 × 0.10 = $3,600 in taxes. Therefore, Nick's net pay for Job B is $36,000 - $3,600 = $32,400. The difference in net pay between the two jobs is $32,400 - $24,000 = $8,400. Therefore, Nick will make $8,400 more at the job with a higher net pay rate.
8400
The answer is: 8400.

Grade

gold 8400 · predicted 8400 · correct

Figure 12 remains in the early-to-middle band of Figure 4(b). Block $\ell = 4$ is the mode (90/301 tokens); the input state $\ell = 0$ and blocks $\ell \in \{10, 11, 18\}$ form the next tier, especially on numeric subwords and the boxed `#### 8400` line. Later blocks ($\ell \gtrsim 20$) stay weak through both tax computations and the final difference, so the Qwen2.5-7B math adapter does not migrate to the general-domain late-depth regime as the answer is emitted.

Case B Mathematical SFT / Gemma-7B

GSM8K · index 331 · Gemma-7B · 0-shot greedy · numerical accuracy

Prompt

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:
Nick is choosing between two jobs. Job A pays $15 an hour for 2000 hours a year, and is in a state with a 20% total tax rate. Job B pays $42,000 a year and is in a state that charges $6,000 in property tax and a 10% tax rate on net income after property tax. How much more money will Nick make at the job with a higher net pay rate, compared to the other job?

Response: Let's think step by step.

Completion

Job A pays $15 an hour for 2000 hours a year, so the total earnings for Job A is $15 x 2000 = $30,000. The state with Job A has a 20% total tax rate, so the total tax for Job A is $30,000 x 20% = $6,000. The net pay for Job A is $30,000 - $6,000 = $24,000. Job B pays $42,000 a year and the state with Job B charges $6,000 in property tax, so the net pay for Job B is $42,000 - $6,000 = $36,000. The state with

Job B has a 10% tax rate on net income after property tax, so the total tax for Job B is $36,000 x 10% = $3,600. The net pay for Job B is $36,000 - $3,600 = $32,400. The difference in net pay between the two jobs is $32,400 - $24,000 = $8,400. Therefore, Nick will make $8,400 more at the job with a higher net pay rate compared to the other job.
8400
The answer is: 8400.

Grade

gold 8400 · predicted 8400 · correct

Figure 13 does not follow the early-to-middle band of the Qwen2.5-7B math adapter. Block $\ell = 23$ is the mode (132/333 tokens), and mean mass is highest at $\ell \in \{23, 21, 22, 26\}$. Early blocks rarely win a column. On this Gemma checkpoint, mathematical SFT therefore occupies a late-decoder band closer to Figure 4(a) than to Figure 4(b).

Case B Mathematical SFT / Llama-3.1-8B

GSM8K · index 331 · Llama-3.1-8B · 0-shot greedy · numerical accuracy

Prompt

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:
Nick is choosing between two jobs. Job A pays $15 an hour for 2000 hours a year, and is in a state with a 20% total tax rate. Job B pays $42,000 a year and is in a state that charges $6,000 in property tax and a 10% tax rate on net income after property tax. How much more money will Nick make at the job with a higher net pay rate, compared to the other job?

Response: Let's think step by step.

Completion

For Job A, Nick will earn $15 x 2000 = $30,000 per year. After taxes, Nick will earn $30,000 - ($30,000 x 20%) = $30,000 - $6,000 = $24,000 per year. For Job B, Nick will earn $42,000 per year. After property tax, Nick will earn $42,000 - $6,000 = $36,000 per year. After taxes, Nick will earn $36,000 - ($36,000 x 10%) = $36,000 - $3,600 = $32,400 per year. To find out how much more money Nick will make at Job B compared to Job A, we subtract the net pay from Job A from the net pay from Job B. $32,400 - $24,000 = $8,400. Therefore, Nick will make $8,400 more at the job with a higher net pay rate.
8400
The answer is: 8400.

Grade

gold 8400 · predicted 8400 · correct

Figure 14 is more diffuse than either Qwen2.5-7B or Gemma-7B on this item. Llama-3.1-8B has 32 decoder blocks; mean mass is highest at $\ell \in \{23, 15, 24, 16\}$, and the argmax spreads from the middle of the stack into the late twenties. The mode is not the Qwen2.5-7B early block $\ell = 4$, nor Gemma's concentrated $\ell = 23$. Taken together, the three mathematical traces show that preferring shallow depth on GSM8K is a property of the Qwen2.5-7B math adapter rather than a domain-universal rule: Gemma remains late, and Llama occupies a broader mid-to-late band, even though all three adapters answer the same item correctly.

Code generation. HumanEval 124 (`valid_date`), 0-shot function completion, pass@1 = 1.

Case C Code SFT / Qwen2.5-7B

HumanEval / 124 · `valid_date` · Qwen2.5-7B · 0-shot greedy · pass@1

Prompt

```
def valid_date(date):
    """You have to write a function which validates a given date string and
```

returns True if the date is valid otherwise False.
The date is valid if all of the following rules are satisfied:
1. The date string is not empty.
2. The number of days is not less than 1 or higher than 31 days for months
1,3,5,7,8,10,12. And the number of days is not less than 1 or higher
than 30 days for months 4,6,9,11. And, the number of days is not less
than 1 or higher than 29 for the month 2.
3. The months should not be less than 1 or higher than 12.
4. The date should be in the format: mm−dd−yyyy
valid_date('03−11−2000') => True
valid_date('15−01−2012') => False
valid_date('04−0−2040') => False
valid_date('06−04−2020') => True
valid_date('06/04/2020') => False
"""

Completion

```
if date == "":
    return False
if date.count("−") != 2:
    return False
month, day, year = date.split("−")
if not month.isdigit() or not day.isdigit() or not year.isdigit():
    return False
month, day, year = int(month), int(day), int(year)
if month < 1 or month > 12:
    return False
if day < 1 or day > 31:
    return False
if month in [4, 6, 9, 11] and day > 30:
    return False
if month == 2 and day > 29:
    return False
return True
```

Grade

pass@1 = 1

Figure 15 is more diffuse. Mean mass is highest at $\ell \in \{8, 9, 11, 15\}$, but later blocks remain competitive, and the peak a_l is lower than in the two natural-language cases. Control tokens (`if`, `return`, `False`) and identifiers (`month`, `day`, `year`) do not share a single layer: the former often peak near $\ell = 8$–11, while comparison literals and Boolean terminals move later. That role-dependent spread matches the main-text observation for code SFT.

Across the three domain records on Qwen2.5-7B, domain identity sets the preferred depth band and token identity modulates the peak inside that band. Lengthening the completion and restoring every generated token does not reverse the domain-level profiles of Figure 4. Repeating the mathematical item on Gemma-7B and Llama-3.1-8B further shows that the preferred band itself is backbone-dependent. These results demonstrate that LLARA equips a frozen pretrained LLM with token-wise attention residuals, so that downstream adaptation can learn an input-dependent readout over block-level residual increments rather than remaining confined to uniform PreNorm accumulation.

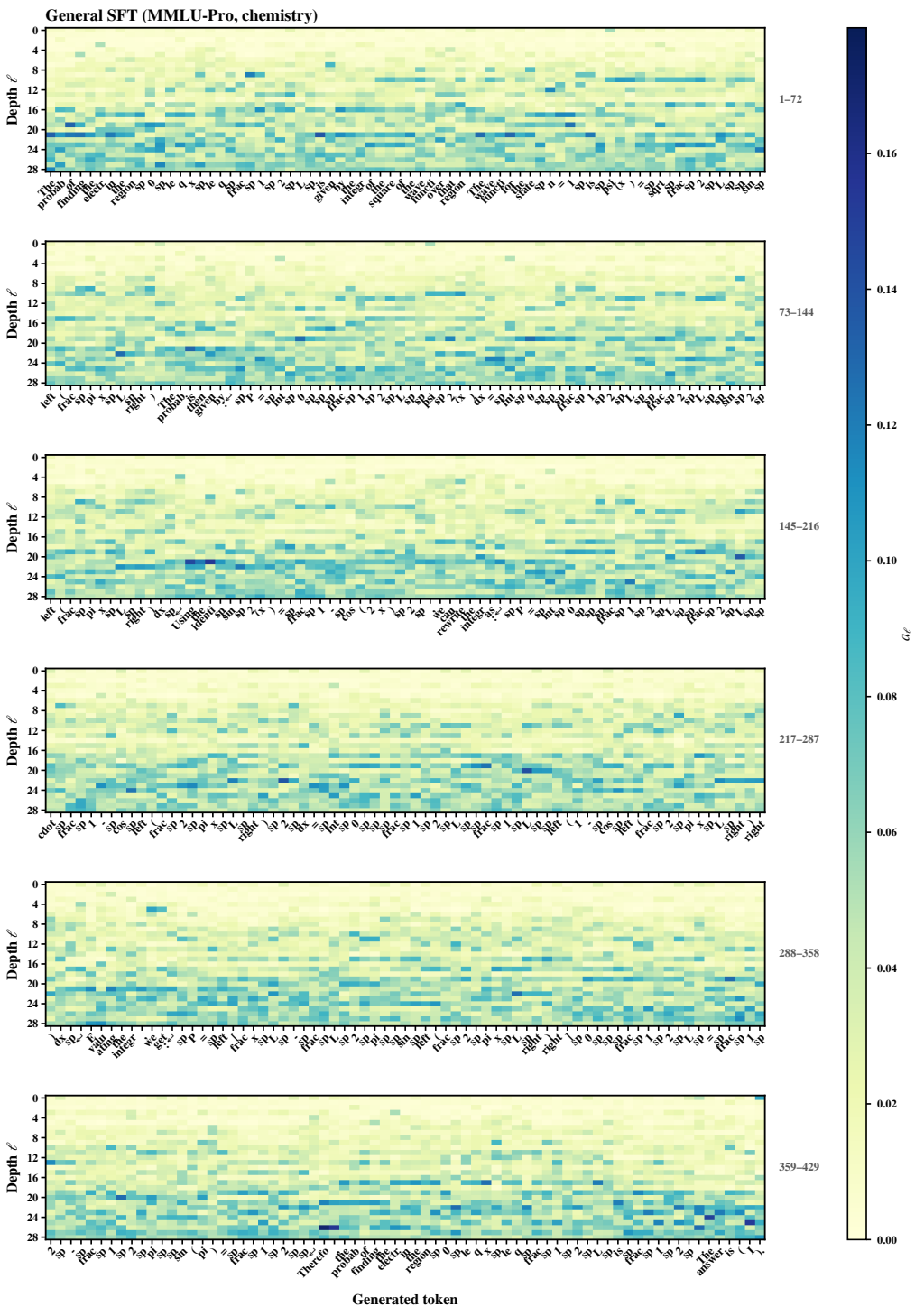


Figure 11: Full completion for Case A on Qwen2.5-7B (429 tokens), split into even strips. Color is a_l on an independent scale. Depth $l = 0$ is the input state.

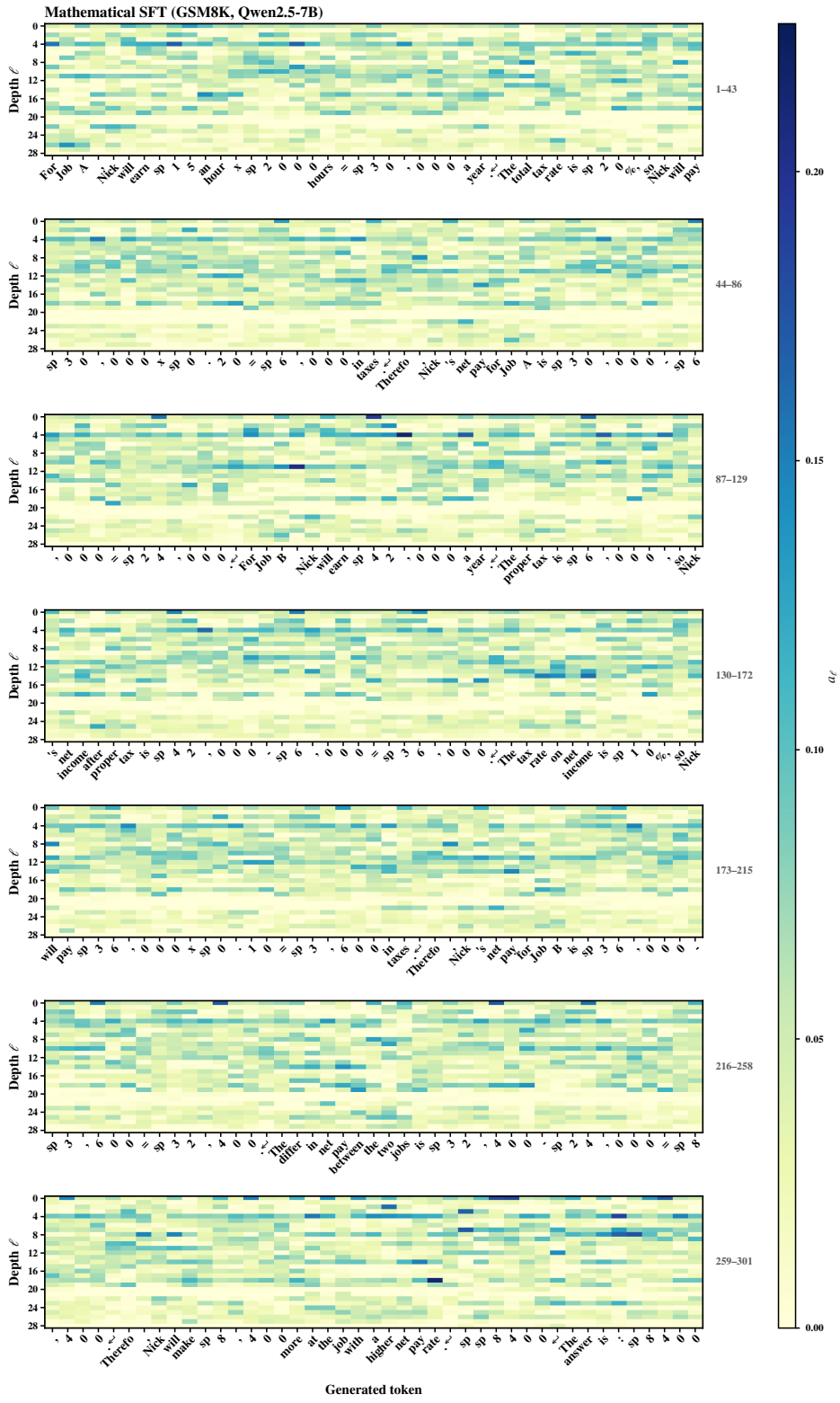


Figure 12: Full completion for Case B on Qwen2.5-7B (301 tokens), split into even strips. Color is a_l on an independent scale.

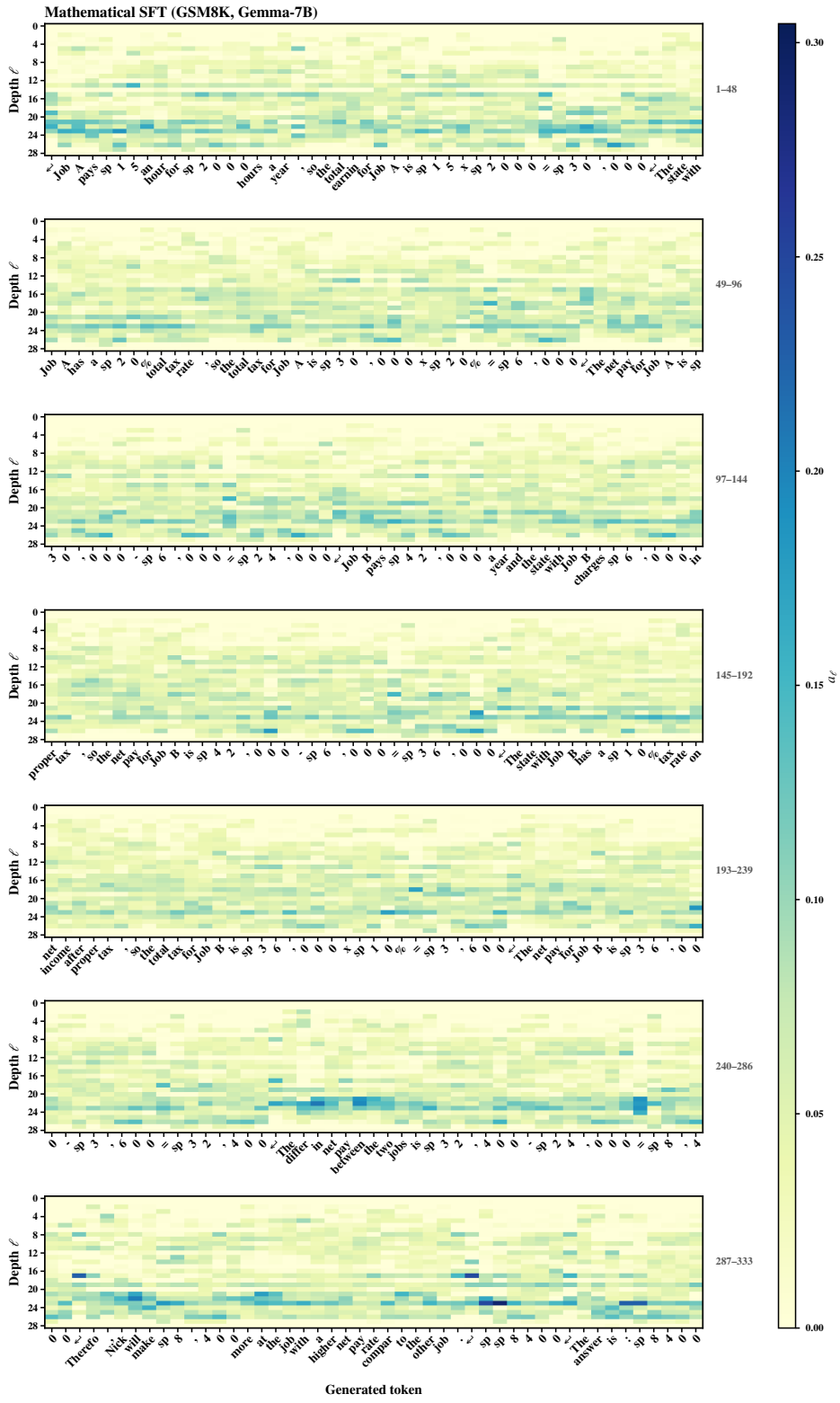


Figure 13: Full completion for Case B on Gemma-7B (333 tokens), split into even strips. Color is a_l on an independent scale.

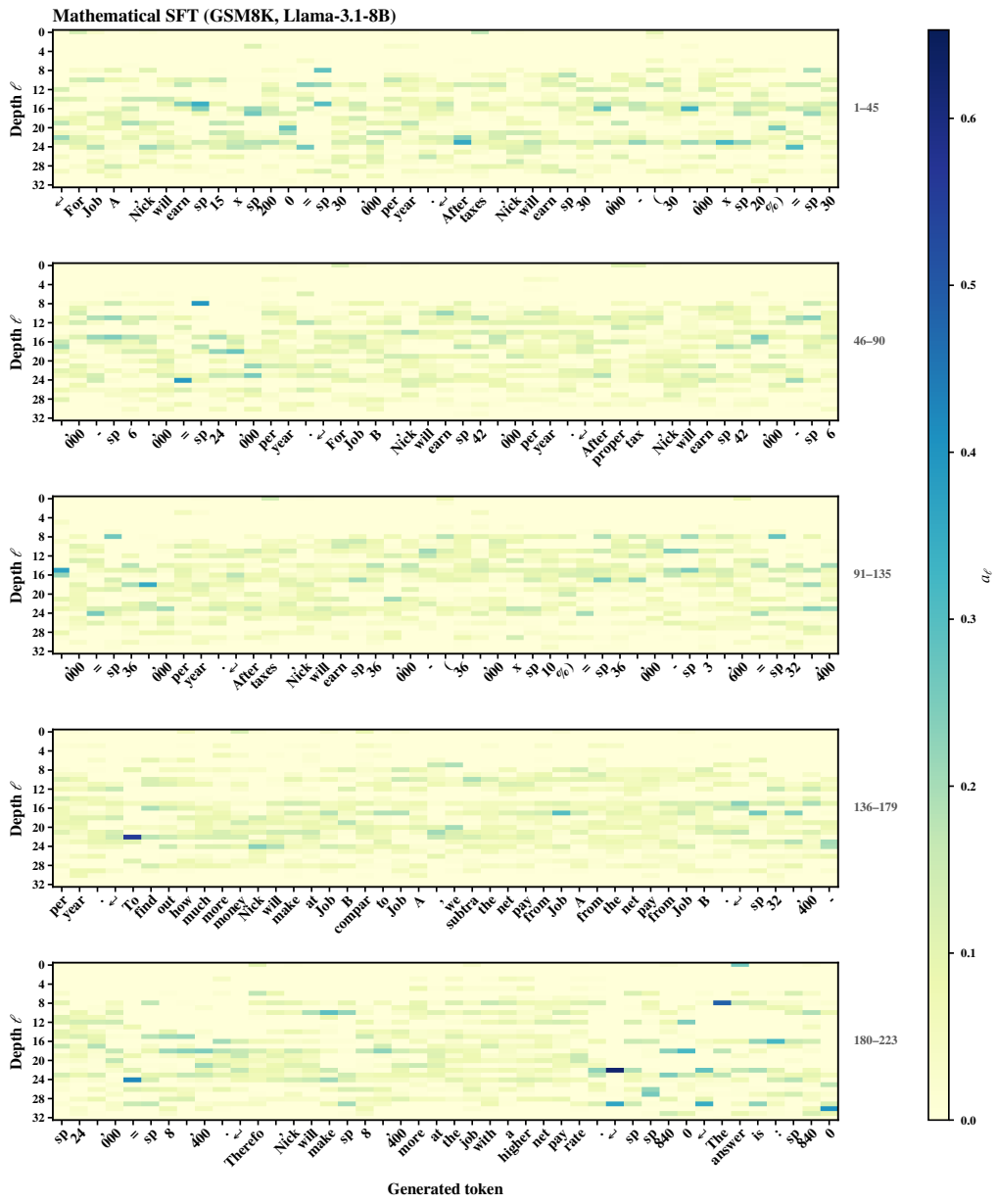


Figure 14: Full completion for Case B on Llama-3.1-8B (223 tokens), split into even strips. Color is a_l on an independent scale. Depth runs to $\ell = 32$ because Llama-3.1-8B has 32 decoder blocks.

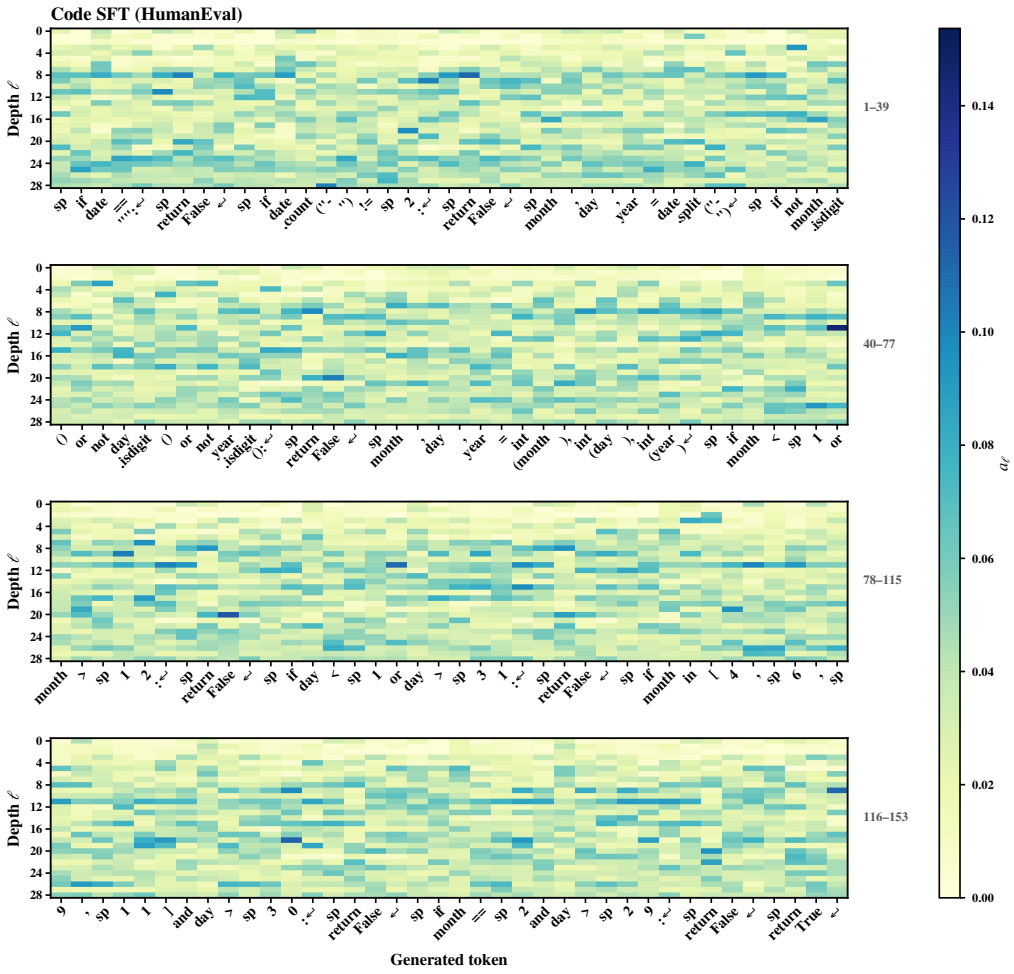


Figure 15: Full completion for Case C on Qwen2.5-7B (153 tokens), split into even strips. Color is a_l on an independent scale.