PRISM TRANSFORMER: PROGRESSIVE HEAD SCHEDULES FOR HIERARCHICAL ATTENTION PROCESSING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-head attention conventionally partitions the hidden dimension equally across all heads at every layer, enforcing an identical representational subspace dimension ($d_h = d_{model}/h$) throughout the model's depth. In this work, we investigate whether this uniform allocation limits the representational capacity available to different stages of a Transformer. To address this, we introduce the Prism Transformer, a novel architectural paradigm that replaces the static, uniform head configuration with a progressive head schedule. By monotonically increasing the head count across layers, the Prism Transformer establishes a local-to-global representational hierarchy: early layers leverage fewer, wider heads to capture complex, local compositional patterns, while deeper layers deploy more, narrower heads to refine these representations into specialized linguistic features. Crucially, this structural shift is parameter-neutral and theoretically compute-neutral, preserving identical weight matrices and theoretical FLOP budgets as the standard Transformer. Across three model scales (124M, 354M, and 757M), the Prism Transformer consistently achieves lower validation loss than uniform baselines while maintaining identical measured training throughput on our 8× H100 GPU setup, alongside improved performance on downstream zero-shot benchmarks including PIQA, HellaSwag, ARC-Easy, and WinoGrande. Our findings suggest that non-uniform subspace allocation can improve the use of model capacity within the standard Transformer parameter and FLOP budget. Our code is available at `https://anonymous.4open.science/r/PrismTransformer-593A/`

1 INTRODUCTION

The multi-head attention (MHA) mechanism (Vaswani et al., 2017) is the defining component of the Transformer architecture. By projecting queries, keys, and values into h independent subspaces of dimension $d_h = d_{\text{model}}/h$, MHA enables the model to simultaneously attend to multiple distinct representational patterns. This division of the hidden dimension has been held constant across all layers since the original Transformer: every layer in a standard decoder uses the identical number of heads, and therefore operates within the exact same per-head dimension.

This uniform allocation embeds a strong, implicit assumption about representation processing: that all layers across a network's depth benefit equally from the same granularity of attention subspaces. We argue that this assumption introduces a fundamental architectural mismatch between what early layers require and what a uniform schedule forces them to execute.

The problem with uniform allocation Early transformer layers are responsible for integrating raw token embeddings into high-level representations that capture complex, local compositional semantic structures. Encoding these intricate local patterns effectively requires substantial representational capacity per subspace. Under a standard uniform configuration (e.g., $h = 12$ heads over $d_{model} = 768$), each early-layer head is restricted to a narrow $d_h = 64$ dimensions. Due to this restricted dimensional space, individual heads have limited capacity to model complex, high-dimensional local contextual features.

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.

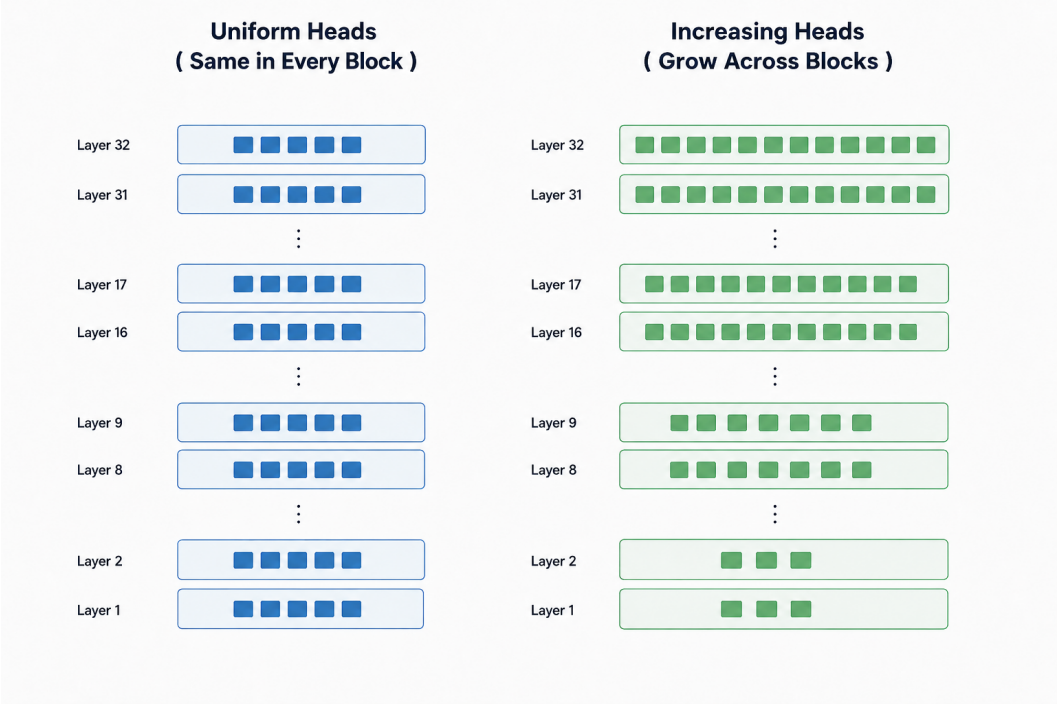


Figure 1: **Head schedule visualization** Comparison of attention head allocations across layers. (Left) Standard baseline featuring uniform heads (a flat schedule across all blocks). (Right) The Prism Transformer featuring an increasing head schedule (a progressive staircase growth across blocks). By expanding the head count in deeper layers, the Prism Transformer implicitly creates a monotonically decreasing per-head dimension (d_h) trajectory.

As representations progress through the network, their structural needs evolve. Mid-network layers serve as the primary engine for global contextual integration, aggregating these dense local features across long-range sequence dependencies. Finally, late transformer layers shift away from broad integration toward specialized feature decomposition, extracting and isolating specific fine-grained syntactic, semantic, or task-oriented signals. This downstream refinement objective is naturally well-suited to a high density of narrow, focused subspaces operating in parallel. By forcing every stage, including local composition, global integration, and fine-grained refinement, into identical structural configurations, the conventional uniform schedule ignores the natural local-to-global trajectory of the network.

The Prism Transformer We propose to resolve this structural bottleneck by replacing the rigid, uniform configuration with a progressive head schedule: a non-decreasing sequence $(h_1, h_2, \ldots, h_L)$ where h_l denotes the head count of layer l. Early layers utilize a small head count, granting each individual head an exceptionally wide subspace ($d_h^{(l)} = d_{\text{model}}/h_l$ is large when h_l is small). The head count then increases monotonically across the network's depth, converging to the standard baseline configuration in later layers.

The per-head dimension narrows across the progressive phase, which gives the architecture its name. This design has two structural properties. First, it is parameter-neutral: because the MHA projection matrices (W_Q, W_K, W_V, W_O) keep their standard $\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ shapes, changing the number of heads does not change the parameter count. Second, it is compute-neutral: the matrix-multiplication FLOPs of attention do not depend on the head count (Section 2.4).

Figure 1 illustrates the contrast between a uniform and a Prism schedule.

Contributions Our primary contributions are as follows:

- **Prism Transformer:** We propose a progressive head schedule for decoder-only Transformers that establishes a robust local-to-global representational hierarchy at zero parameter or compute overhead.
- **Consistent Gains Across Scales:** Across three model scales (124M, 354M, 757M), the Prism Transformer achieves lower validation loss than the uniform baselines at identical training compute. (Table 2).
- **Attention-Distance Analysis:** We conduct a detailed per-layer attention distance analysis, empirically demonstrating that the progressive schedule restructures the network's attention profile. Early wide-head layers attend more locally than the baseline, and the layers after the schedule boundary attend more globally (Section 4).
- **Downstream Benchmark Parity and Improvements:** Evaluation across zero-shot language benchmarks (PIQA, HellaSwag, ARC-Easy, and WinoGrande) confirms that our progressive schedule preserves or explicitly improves downstream task-level accuracy. (Table 3, Figure 2).
- **Head schedule design principles:** Through systematic structural ablations, we isolate the specific geometric criteria, such as dimension transition smoothness and block consolidation, that govern schedule effectiveness. We formalize these findings into a compact set of transferable design rules that generalize consistently across all evaluated parameter scales (Section 2).

2 The Prism Transformer

2.1 Mathematical Formulation of Progressive Head Schedules

Let an L-layer Transformer possess a constant model dimension d_{model}. A progressive head schedule is defined as a non-decreasing integer sequence $\mathcal{S} = (h_1, h_2, \ldots, h_L)$ where h_l denotes the allocation of attention heads at layer l, subject to the following structural constraints:

(i) Divisibility: $h_l \mid d_{\text{model}}$ for all $l \in \{1, \ldots, L\}$ (head count divides model dimension),

(ii) Monotonicity: $h_1 \leq h_2 \leq \cdots \leq h_L$ (monotonically non-decreasing),

(iii) Boundary Convergence: $h_L = h_{\text{base}}$ (final layers match the baseline head count)

2.2 Empirical Design Guidelines

While the constraints in Section 2.1 define the space of valid schedules, optimization within this space requires tuning. Through systematic structural ablations on candidate shapes (detailed in Appendix D), we formalize two vital design principles that dictate schedule effectiveness:

(i) Staircase Smoothness: Abrupt, single-step transitions degrade performance. Optimal schedules utilize localized, multi-layer consolidation phases (e.g., maintaining a specific head dimension for at least 2 to 4 consecutive layers) to stabilize representations before altering granularity.

(ii) Baseline Preservation Phase: To ensure robust semantic convergence, at least half of the network's total layers ($L/2$) should be dedicated to the baseline head count h_{base}.

2.3 Hardware Alignment & Complexity

A primary advantage of the Prism Transformer is its zero-overhead integration into modern compute clusters. Altering head counts does not touch the parameter space: because the projection matrices ($W_Q, W_K, W_V, W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$) maintain their standard uniform shapes, total parameters remain strictly invariant.

We also restrict schedules to head dimensions that fused attention kernels execute natively. These kernels compute attention in tiles that hold each query block's full head dimension in on-chip memory, and are built for head dimensions that are multiples of 32 up to a maximum of 256. Every head dimension in our main schedules ($d_h^{(l)} \in \{256, 128, 96, 64\}$) satisfies both conditions, so each layer runs on the same fused kernel path as the baseline. Head dimensions above 256 fall back to a slower attention path and measurably reduce throughput (Appendix D).

2.4 Parameter and Compute Neutrality

Parameter Invariance The parameter budget of a standard multi-head attention block at layer l is determined entirely by the projection operators for queries, keys, values, and output mapping: $W_Q, W_K, W_V, W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$. The total parameter count for a single layer's MHA module is formalized as:

$$N_{\text{params}}^{(l)} = 4d_{\text{model}}^2 + 4d_{\text{model}} \cdot \mathbb{I}_{\text{bias}}$$

where $\mathbb{I}_{\text{bias}} \in \{0, 1\}$ represents an indicator variable for the presence of a bias vector. Because this allocation depends strictly on the global model dimension d_{model}, it remains fundamentally invariant to the layer-specific head count h_l. Prism therefore reallocates attention capacity across depth without changing the parameter count.

Computational Complexity We analyze the exact computational complexity for a forward pass of a single attention layer over a sequence of length T, omitting the batch dimension and counting one multiply-add operation as 2 FLOPs. The dominant matrix multiplications decompose into four primary operations:

(i) **QKV projections:** computing XW_Q, XW_K, and XW_V requires $6\,T d_{\text{model}}^2$ FLOPs.

(ii) **Attention scores:** computing $QK^\top$ for all h_l heads requires $2\,T^2 h_l d_h^{(l)} = 2\,T^2 d_{\text{model}}$ FLOPs.

(iii) **Value aggregation:** computing AV for all h_l heads requires $2\,T^2 h_l d_h^{(l)} = 2\,T^2 d_{\text{model}}$ FLOPs.

(iv) **Output projection:** computing YW_O requires $2\,T d_{\text{model}}^2$ FLOPs.

Summing these terms yields an exact matrix-multiplication FLOP count of $8\,T d_{\text{model}}^2 + 4\,T^2 d_{\text{model}}$, which depends strictly on T and d_{model} and is invariant to head partition layout h_l. Causal masking scales terms (ii) and (iii) down by a factor of 2 identically regardless of h_l. Backward pass computations mirror these exact matrix dimensions, maintaining operational invariance during training.

The only head-dependent overhead arises from lower-order element-wise operations (softmax normalization and scale factor application), contributing $\mathcal{O}(h_l T^2)$ FLOPs. Expressed relative to the matrix-multiplication terms (ii) and (iii), this overhead constitutes a negligible fraction of order $\mathcal{O}(1/d_h^{(l)})$. Because $h_l \leq h_{\text{base}}$ at every layer, Prism's lower-order cost is bounded above by the baseline's. Prism therefore never adds FLOPs relative to the baseline: the matrix-multiplication FLOPs are identical, and the head-dependent lower-order FLOPs are equal or smaller.

Hardware alignment Because every head dimension in the schedule is supported natively by the fused attention kernel (Section 2.3), Prism matches the measured training throughput and wall-clock time of the uniform baseline at all three scales (Table 2).

Inference cost The same accounting applies to autoregressive decoding. Each layer caches keys and values of total width d_{model} per token regardless of h_l. The KV cache therefore holds $2\,L\,T\,d_{\text{model}}$ values, as in the baseline, and the per-token decoding FLOPs are likewise unchanged.

3 Experiments

3.1 Experimental Setup

Hardware and Core Implementation All training runs are executed on a distributed cluster consisting of 8× NVIDIA H100 (80GB SXM5) GPUs interconnected via NVLink. Models are built in PyTorch by extending the NanoGPT framework (Karpathy, 2022) with Rotary Positional Embeddings (RoPE) (Su et al., 2021) and SwiGLU activations (Shazeer, 2020). Architectural adjustments are completely isolated to the attention layer split dimensions; all other macro-parameters (e.g., hidden states, layer depths, optimizer choices) remain identical across baseline and experimental setup.

Table 1: Model configurations and training hyperparameters. h_{base} denotes the baseline (uniform) head count; the Prism schedule uses h_{base} only in later layers (see Appendix C). All other architectural hyperparameters are identical across the baseline and Prism variants.

Model	n_{params}	n_{layers}	d_{model}	h_{base}	Batch Size	LR	Training Tokens
Small	124M	12	768	12	0.25M	6.0×10^{-4}	2.88B
Medium	354M	24	1024	16	0.25M	3.0×10^{-4}	10.49B
Large	757M	24	1536	16	0.50M	2.5×10^{-4}	20.97B

Table 2: Validation performance on FineWeb and training efficiency on an $8\times$ NVIDIA H100 cluster. Validation loss is mean $\pm$ standard deviation across three independent seeds. Throughput (aggregate over 8 GPUs) and wall-clock time are means across the three seeds; throughput is rounded to the nearest thousand tokens per second and wall-clock time to the nearest hundredth of an hour. Bold indicates the lower validation loss at each scale.

Params	**Model**	**Val Loss**	**Tokens/sec**	**Wall-clock (hrs)**
124M	Baseline Uniform	3.3748 ± 0.0027	2875K	0.31
	Prism Transformer	$\mathbf{3.3604 \pm 0.0010}$	2875K	0.31
354M	Baseline Uniform	2.9944 ± 0.0015	1025K	3.03
	Prism Transformer	$\mathbf{2.9873 \pm 0.0020}$	1025K	3.03
757M	Baseline Uniform	2.8106 ± 0.0002	656K	9.09
	Prism Transformer	$\mathbf{2.8022 \pm 0.0013}$	656K	9.09

Dataset and Tokenization Models are pre-trained on the FineWeb dataset (Penedo et al., 2024), tokenized with the GPT-2 BPE tokenizer. Following the Chinchilla scaling law (Hoffmann et al., 2022), we train each model for 25 to 30 tokens per parameter, slightly exceeding the compute-optimal ratio to ensure all variants reach sufficient convergence for a robust comparison.

Training Configuration Training uses mixed-precision `bfloat16` under Distributed Data Parallel (DDP) with a fixed context length of 1024 tokens. The AdamW optimizer is configured with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient clipping at 1.0. A cosine decay learning rate schedule with linear warmup over the first 2.5% of training tokens is used throughout. Architectural specifications and hyperparameters are summarized in Table 1.

3.2 Training Results

To evaluate the fundamental architectural impact of progressive head schedules, we compare the pre-training performance of the Prism Transformer against uniform baseline configurations across three distinct parameter scales. To account for variation across random initializations, all configurations are trained with three independent random seeds. We report the empirical mean and sample standard deviation ($\mu \pm \sigma$) calculated at the terminal training checkpoints.

As shown in Table 2, the Prism Transformer consistently achieves lower validation loss across all scales. Importantly, this improvement is achieved purely through schedule redistribution; the model has exactly the same training token budgets, parameter counts, and total FLOP allocations as the standard architecture, but uses it more efficiently by aligning the representational granularity with layer depth.

3.3 Downstream Benchmark Evaluation

We evaluated our final model checkpoints on a broad suite of zero-shot downstream tasks using the LM Evaluation Harness (Gao et al., 2024). To ensure statistical validity, we evaluated all three independent training seeds for both the uniform baselines and the Prism configurations across multiple foundational benchmarks: PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), ARC-

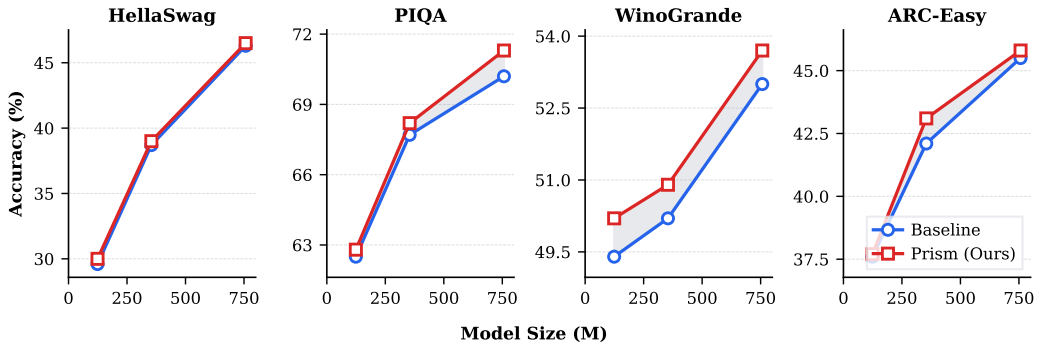


Figure 2: **Zero-shot benchmark performance across model scales.** All comparisons are evaluated at identical token milestones. The Prism Transformer achieves comparable or superior accuracy to the baseline across all tasks.

Easy (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2019), BLiMP (Warstadt et al., 2020), and WikiText (Merity et al., 2017). We report the empirical mean accuracy (μ) across these runs, with full tabulated results provided in Appendix A (Table 3).

As shown in Figure 2, the Prism Transformer consistently preserves or explicitly improves downstream task-level accuracy across all evaluated model sizes. This demonstrates that expanding the early-layer subspace width provides the network with the necessary representational bandwidth to encode more robust, generalizable features that directly benefit contextual knowledge retrieval and structural common-sense reasoning.

3.4 Hardware and System Training Throughput

To verify the system efficiency of the Prism Transformer on modern hardware clusters, we record wall-clock training durations and raw system execution throughput metrics across our distributed environments. We evaluate device metrics across all three independent training seeds to isolate hardware variations, reporting the empirical mean execution speed (tokens per second) and total accumulated wall-clock runtime.

The hardware benchmarks in Table 2 show that Prism matches the baseline's training throughput and wall-clock time at all three scales. This follows from the head-dimension constraint in Section 2.3: every layer uses a head dimension within the native range of fused attention kernels, so no layer falls back to a slower attention path.

3.5 Scaling Experiments

To evaluate the robustness and predictability of progressive head schedules under compute and parameter scaling, we evaluate the Prism Transformer against uniform baselines across three distinct model capacities: **Small (124M)**, **Medium (354M)**, and **Large (757M)**. All model pairs are trained from scratch under identical hyperparameter configurations, dataset mixtures, and token budgets to isolate the structural impact of the head schedule (see Appendix C for exact schedule details).

The pretraining scaling trajectories illustrated in Figure 3 yield several key insights:

- **Consistent Scaling Advantage:** Across all three parameter scales, the Prism Transformer consistently achieves a lower validation cross-entropy loss compared to its uniform baseline counterpart (Figure 3b).
- **Evolution of the gap during training:** In the Large model (Figure 3a), the gap is noisy early in training, and the baseline is ahead below roughly 0.8B tokens. Between 1B and 10B tokens, Prism's advantage grows steadily, and beyond 10B tokens it stabilizes near -0.008 while the seed variance band narrows. The Medium model exhibits a similar trajectory with a crossover around 1.8B tokens, whereas the Small model maintains an advantage from the earliest evaluation (Appendix B, Figure 4).

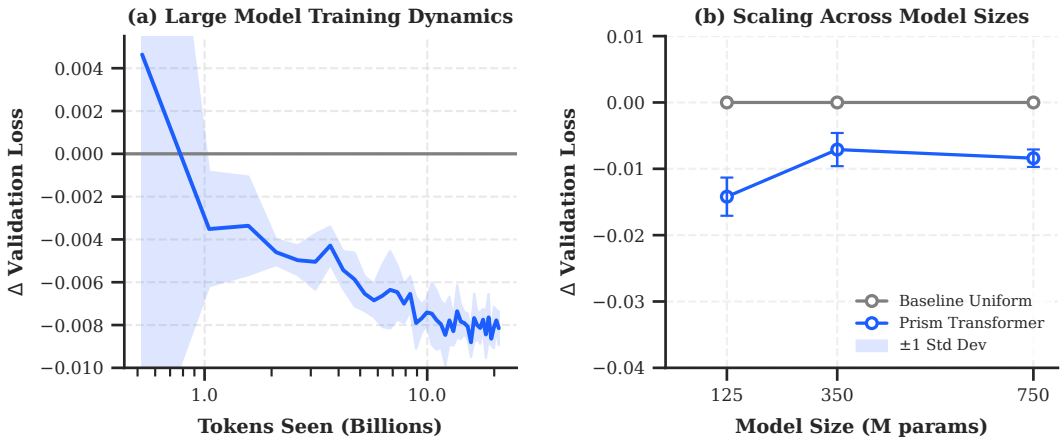


Figure 3: **Scaling properties of Prism Transformer compared to Baseline Uniform.** (a) Training Dynamics. Solid line depicts the mean validation loss gap across three random seeds for the large model, with the shaded region denoting ±1 standard deviation. A negative gap indicates Prism outperforms the baseline. (b) Mean validation loss gap at the end of training for each model size (124M, 354M, and 757M), with error bars showing ± 1 standard deviation propagated across seeds.

4 Analysis

4.1 Attention Distance Reveals a Sharpened Local-to-Global Gradient

To probe how the Prism schedule systematically reorganizes sequence context processing, we track the mean attention distance at every layer for both architectures. For layer l and head i, the attention-weighted mean token distance of a query at position t is $\sum_{s=1}^{t} A_{t,s}^{(l,i)} |t-s|$, where $A_{t,s}^{(l,i)} = \mathrm{Softmax}\big(q_t^{(l,i)} K^{(l,i)\top} / \sqrt{d_h^{(l)}}\big)_s$ are the causal attention weights ($A_{t,s}^{(l,i)} = 0$ for $s > t$ by construction). We report the per-layer distance metric $\mathcal{D}_l$ by averaging across heads, query positions in the last 50% of the context window, and evaluation sequences:

$$\mathcal{D}_l = \frac{1}{h_l} \sum_{i=1}^{h_l} \frac{1}{|\mathcal{Q}|} \sum_{t \in \mathcal{Q}} \mathbb{E}_x \left[\sum_{s=1}^{t} A_{t,s}^{(l,i)} |t-s| \right], \quad \mathcal{Q} = \{\lfloor T/2 \rfloor + 1, \ldots, T\}, \tag{1}$$

where $\mathbb{E}_x$ denotes the empirical expectation over validation sequences. Restricting $\mathcal{Q}$ to positions $t \geq \lfloor T/2 \rfloor + 1 = 513$ (for $T = 1024$) ensures that every query has at least 512 tokens of available context, eliminating the artifact in which early tokens are forced to attend locally due to a short causal history. Because the distance is linear in the attention weights, the head average in equation 1 equals the distance computed on the head-averaged attention map (Vig & Belinkov, 2019; Raghu et al., 2021), so $\mathcal{D}_l$ is independent of both h_l and d_{head}, enabling a direct comparison across schedules with different head counts. All values are in absolute token units, averaged over three independent seeds ($n = 3$).

Figure 5 shows the resulting profiles across all three model scales. The profiles share three features. In the early layers, where the Prism schedule allocates wider subspaces, $\mathcal{D}_l$ is substantially lower than the baseline ($\Delta\mathcal{D}_l < 0$), indicating more local attention. This reverses in the mid-network layers, where $\Delta\mathcal{D}_l$ peaks positively, precisely near the point where the progressive ramp completes. In the final layers, the two profiles converge to similar attention distances. The alignment of the sign change with the schedule boundary at all three scales is consistent with the reorganization being driven by the head allocation. This analysis characterizes how the schedule changes attention behavior; our performance claims rest on the results in Section 3.

4.2 Why Does the Progressive Schedule Win?

Combining the attention-distance analysis (Section 4.1) with the progressive head trajectory (Section 2), we offer the following interpretation of the Prism Transformer's advantage. The attention-

distance measurements are empirical; the account below provides our interpretation, framing non-empirical steps as explicit hypotheses.

(1) Wide early heads build rich local features By allocating a small number of heads in the first layers, each individual head operates within an exceptionally wide 256-dimensional subspace ($d_h = 256$), and these layers attend more locally than the baseline (negative $\Delta\mathcal{D}_l$ in Figure 5). We hypothesize that wider heads let early layers form richer representations of local context than narrow heads spread across many positions.

(2) Later layers compensate with longer-range integration. The positive peak in $\Delta\mathcal{D}_l$ occurs in layers where Prism and the baseline use the same head count, so it cannot come from the head configuration of those layers. Instead, it reflects how these layers respond to different inputs. Because Prism's early layers attend more locally than the baseline's, less long-range information has been aggregated by the time representations reach the baseline-configured layers, and these layers attend further to make up for it. Under this account, Prism shifts global integration to later layers rather than removing it, and those layers operate on inputs that already contain *richer local features*.

(3) Late layers behave similarly in both models. In the final layers, the attention distances of the two models converge. This suggests that both perform a similar kind of processing at this stage, operating on representations that were assembled differently in earlier layers.

This account predicts that the advantage is structural rather than an optimization artifact, meaning it should persist throughout the entire training trajectory. The token scaling curves in Section 3.5 (Figure 3) strongly validate this hypothesis: after an early crossover, the Prism Transformer maintains a lower validation loss at every token milestone.

5 Comparison with Related Work

Head Redundancy and Capacity Many attention heads in trained Transformers can be pruned with little loss in accuracy (Michel et al., 2019; Voita et al., 2019), and tying the head dimension to d_{model}/h creates a low-rank bottleneck that limits what each head can express (Bhojanapalli et al., 2020). Prior work addresses these issues after training through pruning, or by changing the head size uniformly across layers. Prism instead changes the allocation across depth before training: early layers use fewer, wider heads, which relaxes the rank constraint where we hypothesize it matters most, while later layers keep the standard configuration.

Depth and Layer Specialization Probing studies show that Transformer layers form a processing hierarchy, with earlier layers encoding surface-level and syntactic information and later layers encoding more abstract semantic information (Tenney et al., 2019). Several methods exploit this by varying computation or parameters across depth, including structured layer dropout (Fan et al., 2020), early exit, and variable-width parameter allocation (Wu et al., 2026). The closest prior designs to ours allocate capacity non-uniformly across depth by design. DeLighT (Mehta et al., 2020) uses smaller blocks near the input and larger blocks near the output, and OpenELM (Mehta et al., 2024) applies this layer-wise scaling to standard Transformer layers by increasing the number of attention heads and the feed-forward width with depth. Like Prism, OpenELM uses fewer heads in early layers, but it keeps the head dimension fixed, so early layers have narrower attention projections and fewer parameters than later layers. Prism makes the opposite choice: every layer keeps the same projection shapes and parameter count, and only the head dimension changes. Early layers in OpenELM therefore have less attention capacity than later layers, whereas early layers in Prism have the same capacity split into fewer, wider heads.

Inference-Optimized Head Layouts Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) modify head layouts to alleviate the Key-Value (KV) cache memory bottleneck during inference. They change the ratio of query heads to KV heads within each layer while keeping the head dimension uniform across depth. Prism instead targets representational capacity across depth: it keeps the per-layer ratio of query to KV heads fixed and varies the head dimension from layer to layer. Because the two changes act on different axes, Prism can be combined with GQA, and the combination preserves the KV-cache size of uniform GQA (Appendix E).

6 CONCLUSION

We introduced the Prism Transformer, a simple modification to the standard multi-head attention mechanism that replaces the uniform head schedule with a progressive one. By starting with fewer, wider heads in early layers and gradually increasing the head count across depth, the Prism Transformer creates a local-to-global representational hierarchy that naturally aligns with the functional demands at each layer index.

The modification is parameter-neutral and compute-neutral by construction: the total parameter count and theoretical FLOPs are identical to the uniform baseline. Yet across all three model scales, the Prism Transformer achieves consistently lower validation loss at equivalent training compute budgets and matches or exceeds the baseline performance across nearly every benchmark in Table 3. Furthermore, Our per-layer attention-distance analysis further shows that the schedule reorganizes context processing across depth: early wide-head layers attend more locally than the baseline, and the layers after the schedule boundary attend more globally. We view this result as an instance of a broader principle, namely that architectural inductive biases respecting the natural functional specialization of depth can yield genuine representational improvements at no additional cost. The Prism Transformer demonstrates that, in standard autoregressive language models, the head configuration is a meaningful axis of architectural design that has been under-explored.

Limitations Our evaluation focuses on decoder-only models for causal language modeling at a context length of 1024 up to 757M parameters; encoder architectures, long-context settings, and larger scales are important areas for future work. At these scales, reasoning-heavy benchmarks like MMLU and ARC-Challenge are near chance, limiting our downstream evaluation. Additionally, while the Medium and Large schedules are constructed from our design rules without further tuning, dynamically adaptive, or continuous schedule parameterizations presents a promising avenue to unlock further improvements.

AI USE STATEMENT

In this work, we used generative AI tools to verify and optimize our implementation of the per-head-dimension rotary embedding (RoPE) caching mechanism (Algorithm 1, Appendix F.1) and to assist in generating Figure 1. We have not used generative AI tools to develop the Prism Transformer's theoretical framework or mathematical formulation (Section 2), design the training and ablation experiments (Section 3, Appendix D), or interpret the results (Sections 3–4); this work is entirely our own. Generation of synthetic data, assistance with proofs, translation, dataset cleaning, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for readability and polish the writing. All AI-assisted code changes were manually reviewed and tested by the authors to confirm they preserved the intended behavior, and all AI-assisted text edits were checked against the underlying technical content to ensure no claim, equation, or result was altered or misrepresented. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure the reproducibility of our results. Our code, including the Prism attention implementation and training scripts, is available at the anonymous repository linked in the abstract. The exact progressive head schedules used for all three model scales are given in Table 4 (Appendix C), together with the rule used to derive them; the complete forward-pass algorithm for the Prism attention layer, including the per-head-dimension RoPE caching scheme, is given in Algorithm 1 (Appendix F.1). All model architectures, training hyperparameters, hardware configuration, and dataset/tokenization choices are summarized in Table 1 and Section 3.1. All reported empirical results (Tables 2, 3, 5, and 6, and Figures 2–4) are averaged over three independent training seeds, with standard deviations reported where applicable, and the full ablation configurations underlying our design choices are listed in Table 5 (Appendix D).

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *ArXiv preprint*, abs/2305.13245, 2023. URL https://arxiv.org/abs/2305.13245.

Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 864–873. PMLR, 2020. URL http://proceedings.mlr.press/v119/bhojanapalli20a.html.

Yonatan Bisk, Rowan Zellers, Ronan LeBras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6239.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv preprint*, abs/1803.05457, 2018. URL https://arxiv.org/abs/1803.05457.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024. URL https://zenodo.org/records/12608602.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *ArXiv preprint*, abs/2203.15556, 2022. URL https://arxiv.org/abs/2203.15556.

Andrej Karpathy. nanogpt: The simplest, fastest repository for training/finetuning medium-sized gpts. https://github.com/karpathy/nanoGPT, 2022. GitHub repository. MIT License.

Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Delight: Deep and light-weight transformer. *ArXiv preprint*, abs/2008.00623, 2020. URL https://arxiv.org/abs/2008.00623.

Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, and Mohammad Rastegari. Openelm: An efficient language model family with open training and inference framework. *ArXiv preprint*, abs/2404.14619, 2024. URL https://arxiv.org/abs/2404.14619.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 14014–14024, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *ArXiv preprint*, abs/2108.08810, 2021. URL https://arxiv.org/abs/2108.08810.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *ArXiv preprint*, abs/1907.10641, 2019. URL https://arxiv.org/abs/1907.10641.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *ArXiv preprint*, abs/1911.02150, 2019. URL https://arxiv.org/abs/1911.02150.

Noam Shazeer. Glu variants improve transformer. *ArXiv preprint*, abs/2002.05202, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *ArXiv preprint*, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 63–76, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4808. URL https://aclanthology.org/W19-4808.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://aclanthology.org/2020.tacl-1.25.

Zhaofeng Wu, Oliver Sieberling, Shawn Tan, Rameswar Panda, Yury Polyanskiy, and Yoon Kim. Variable-width transformers. *ArXiv preprint*, abs/2606.18246, 2026. URL https://arxiv.org/abs/2606.18246.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472.

A Downstream Benchmark Performance and Scale Dynamics

We report the complete zero-shot downstream evaluation suite across all three model scales in Table 3. This includes benchmarks evaluating physical common sense (PIQA), common sense reasoning (HellaSwag, WinoGrande), scientific knowledge (ARC-Easy), structural linguistic capability (BLiMP), and language modeling perplexity (WikiText).

Table 3: Zero-shot downstream performance across model scales. Scores are accuracy (%) for all tasks except WikiText, which reports word-level perplexity (lower is better). Values are means over three independent pre-training seeds. Bold indicates the better model at each scale for each task.

Scale	Model	HellaSwag ↑	PIQA ↑	WinoGrande ↑	BLiMP ↑	ARC-E ↑	WikiText ↓
Small	Baseline	29.6	62.5	49.4	79.5	37.6	52.3
	Prism (Ours)	**30.0**	**62.8**	**50.2**	**80.0**	**37.7**	**51.2**
Medium	Baseline	38.7	67.7	50.2	**80.8**	42.1	33.0
	Prism (Ours)	**39.0**	**68.2**	**50.9**	80.6	**43.1**	**32.9**
Large	Baseline	46.3	70.2	53.0	**81.5**	45.5	25.7
	Prism (Ours)	**46.5**	**71.3**	**53.7**	**81.5**	**45.8**	**25.5**

B Training Dynamics at All Scales

Figure 4 shows the validation loss gap between Prism and the uniform baseline over the course of training at all three scales. The behavior differs by scale. In the Small model, Prism is ahead from the first evaluation, and the gap narrows slightly over training. In the Medium and Large models, the baseline is ahead early in training: until about 1.8B tokens in the Medium model, and until about 0.8B tokens in the Large model. After this point, Prism's advantage grows steadily and Prism remains ahead for the rest of training.

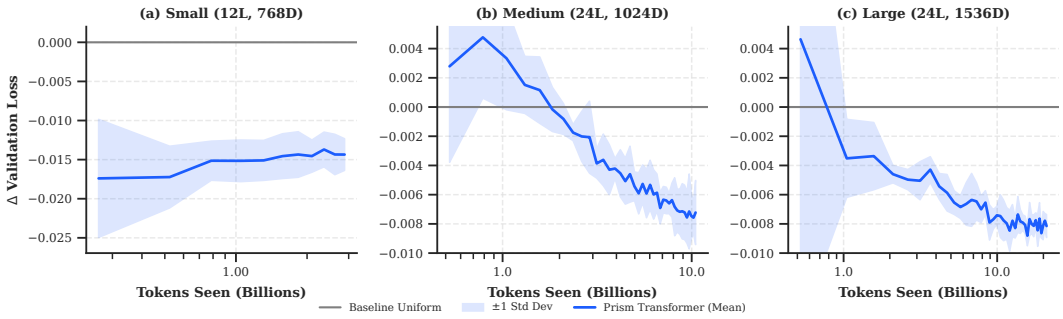


Figure 4: **Validation loss gap during training at all three scales.** Each panel shows the mean gap $\mathcal{L}_{\text{Prism}} - \mathcal{L}_{\text{baseline}}$ across three seeds (solid line), with the shaded region showing ± 1 standard deviation. Negative values indicate that Prism outperforms the baseline. The x-axis is logarithmic.

C Prism Head Schedules by Scale

Table 4 lists the head schedules used in our main experiments. While the Small schedule is selected via the ablation in Appendix D, the Medium and Large schedules are constructed by applying the design rules of Section 2 and hardware constraints of Section 2.3 based on those ablation insights. Because these larger schedules are derived without per-scale search, the performance gains at 354M and 757M parameters highlight the transferability of our design rules.

Note that head counts at each phase must divide d_{model} so that the fused QKV projection can be reshaped into equal heads. We further require $d_h^{(l)}$ to be a multiple of 32 and at most 256, so that every layer stays within the native range of fused attention kernels (Section 2.3). For the Small

Table 4: Progressive head schedules used for the Prism Transformer at each scale. The variable $d_h^{(l)}$ represents the corresponding per-head subspace dimension at layer l.

Scale	d_{model}	L	h_{base}	**Prism Schedule (heads per layer)**	$d_h^{(l)}$ per phase
Small	768	12	12	$(\underbrace{3,3}, \underbrace{6,6}, \underbrace{8,8}, \underbrace{12,12,12,12,12,12})$	256, 128, 96, 64
Medium	1024	24	16	$(\underbrace{4,4,4}, \underbrace{8,8,8}, \underbrace{16,\ldots,16}_{18\times})$	256, 128, 64
Large	1536	24	16	$(\underbrace{6,6,6}, \underbrace{12,12,12}, \underbrace{16,\ldots,16}_{18\times})$	256, 128, 96

configuration ($d_{\text{model}} = 768$), the head counts between 3 and $h_{\text{base}} = 12$ that satisfy these constraints are $\{3, 4, 6, 8, 12\}$, from which we select the progressive sequence $\{3, 6, 8, 12\}$.

For the larger configurations ($d_{\text{model}} \in \{1024, 1536\}$), we choose the initial head count so that every scale starts from the same head dimension, $d_h = 256$ ($768/3$, $1024/4$, and $1536/6$). The Medium configuration then progresses through $\{4, 8, 16\}$ ($d_h = 256, 128, 64$) and the Large configuration through $\{6, 12, 16\}$ ($d_h = 256, 128, 96$), all within this range.

D Ablation Studies

D.1 Schedule Sensitivity at Small Scale

We evaluate ten candidate head schedule configurations at the Small scale (124M parameters, 12 layers) to systematically isolate the architectural properties that dictate representation quality and hardware efficiency. All configurations are pre-trained from scratch using identical compute budgets, optimization hyperparameters, and total parameter allocations as the primary experiments. Table 5 summarizes these structural variants along three primary architectural axes:

- **Initial Subspace Dimension (h_1):** The starting head count in the first layer ($h_1 \in \{2, 3, 4, 6, 12\}$), which controls the maximum initial representational subspace width.
- **Phase Transition Cadence:** The number of unique head count stages (ranging from 2 to 5 phases) deployed across depth prior to arriving at the baseline configuration ($h_{\text{base}} = 12$).
- **Monotonicity and Trajectory:** The direction and smoothness of the head allocation ramp across layers.

Table 5: Ablation study over progressive head schedule configurations at the Small scale ($L = 12$, $d_{\text{model}} = 768$, $h_{\text{base}} = 12$). All variants are trained with identical compute budgets and parameter counts. Reported validation losses and corresponding deltas ($\Delta = L_{baseline} - L_{config}$) represent the empirical mean across three independent training seeds, calculated relative to the uniform baseline. Throughput changes are evaluated on an $8\times$ H100 GPU cluster.

Config	**Head Schedule**	**Description**	**Val Loss ↓**	**Δ vs. Baseline**	**Tok/s**
Config-0	$(12, 12, 12, 12, 12, 12, 8, 8, 6, 6, 3, 3)$	Regressive (inverse)	3.3857	-0.0109	0.0%
Config-1	$(3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3)$	Uniform wide	3.3810	-0.0062	0.0%
Baseline	$(\mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12})$	**Uniform**	3.3748	–	–
Config-2	$(3, 6, 8, 12, 12, 12, 12, 12, 12, 12, 12, 12)$	Linear-like Ramp	3.3691	0.0056	0.0%
Config-3	$(6, 6, 8, 8, 12, 12, 12, 12, 12, 12, 12, 12)$	Higher h_{start}	3.3683	0.0065	0.0%
Config-4	$(4, 4, 8, 8, 12, 12, 12, 12, 12, 12, 12, 12)$	Higher h_{start}	3.3655	0.0093	0.0%
Config-5	$(6, 6, 6, 6, 12, 12, 12, 12, 12, 12, 12, 12)$	2-phase	3.3647	0.0100	0.0%
Config-6	$(2, 4, 6, 8, 12, 12, 12, 12, 12, 12, 12, 12)$	Linear-like Ramp	3.3647	0.0101	−4.3%
Config-7	$(3, 3, 6, 6, 12, 12, 12, 12, 12, 12, 12, 12)$	3-phase	3.3628	0.0120	0.0%
Prism (Ours)	$(\mathbf{3}, \mathbf{3}, \mathbf{6}, \mathbf{6}, \mathbf{8}, \mathbf{8}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12}, \mathbf{12})$	**4-phase smooth**	3.3604	0.0144	0.0%
Config-8	$(2, 2, 4, 4, 8, 8, 12, 12, 12, 12, 12, 12)$	Smooth Ramp	3.3598	0.0150	−8.3%

Key Insights and Structural Properties

1. **Direction, not non-uniformity, drives the gain** To test whether the improvement comes from the local-to-global ordering or simply from breaking depth-wise uniformity, we evaluate two controls under identical parameter and FLOP budgets. Config-0 uses exactly the Prism head counts in reverse order ($12 \to 8 \to 6 \to 3$). It therefore contains the same set of per-layer head dimensions as Prism and differs only in direction. Config-1 assigns wide heads (h = 3, d_h = 256) to every layer. Both controls underperform the uniform baseline (Δ = -0.0109 and -0.0062), while Prism improves on it (Δ = +0.0144). Because Config-0 and Prism share identical per-layer head dimensions, the 0.0253 gap between them isolates the effect of ordering. Config-1 further shows that wide heads are not beneficial in themselves: they help when placed early and hurt when used throughout the network.
2. **The Pareto Frontier of Hardware Alignment:** Config-8 achieves the largest raw validation loss reduction ($\Delta = 0.0150$), but it incurs a substantial hardware throughput penalty of -8.3%. This degradation occurs because an initial head count of 2 forces a per-head dimension of $d_h = 384$, which exceeds the maximum head dimension of 256 supported by commonly used fused attention kernels, so these layers fall back to a slower attention path. Config-6 has the same cause in one layer rather than two, and its penalty (-4.3%) is roughly half as large. The Prism Transformer resolves this trade-off by selecting an initial head count of 3 ($d_h = 256$), capturing over 95% of the maximum optimization gains ($\Delta = 0.0144$) while preserving throughput neutrality (0.0%).
3. **Initial head width (h_1):** Isolating the starting head configurations reveals a clear trend: providing early attention layers with wider per-head dimensions generally improves model performance. Comparing multi-phase variants across identical layer budgets shows that starting with $h_1 = 6$ heads (Config-3, $\Delta = 0.0065$) is surpassed by $h_1 = 4$ (Config-4, $\Delta = 0.0093$), which is further improved by $h_1 = 3$ (Config-7, $\Delta = 0.0120$). This trend supports our core architectural hypothesis that increasing representational capacity in early layers enhances overall processing efficiency.
4. **Staircase Smoothness Dictates Representation Stability:** Abrupt structural shifts underperform smoother transitions across depth. For example, transitioning aggressively from wide heads to baseline dimensions in a single step (Config 2 and Config-6) yields a lower improvement than smoother transitions (Prism and Config-8). Sustaining stable head dimensions across consecutive layer blocks, as seen in the multi-phase layout of the Prism Transformer, provides sufficient depth for the network to steadily compose features at a given representational scale before transitioning to finer attention subspaces.

E Composability with Grouped-Query Attention

As discussed in Section 5, the Prism schedule is orthogonal to the query/key–value ratio manipulations introduced by Grouped-Query Attention (GQA) (Ainslie et al., 2023): Prism redistributes the *per-layer head dimension* across depth, whereas GQA reduces the number of *KV projections* within a layer to shrink the inference-time KV cache. Because the two mechanisms act on independent axes of the attention layout, they should compose without interference. To verify this empirically, we evaluate all four combinations of {uniform, progressive} head schedules with {full MHA, GQA} attention at the Small (124M) scale.

Setup: The two attention families do *not* share a parameter count: GQA shrinks the K and V projections by sharing them across query heads, reducing the attention parameter budget relative to full MHA. Our parameter- and compute-neutrality claims therefore apply *within* each attention family, not across them. Accordingly, we hold the parameter count fixed *within* each family and compare each progressive schedule against the uniform baseline of its *own* family. All four configurations are otherwise trained from scratch under identical compute budgets, optimizer settings, and token allocations as the primary Small-scale experiments (Section 3.1). The uniform GQA variant uses 12 query heads and 6 KV heads in every layer, a query-to-KV ratio of 2. When the Prism schedule is applied on top of GQA, the number of KV heads scales with the per-layer query-head count (2, 2, 4, 4, 6, 6, . . .), so the ratio stays at 2 in every layer and the KV-cache size matches the uniform GQA variant. All results are reported as mean $\pm$ standard deviation over three independent seeds.

Schedule choice under GQA: The Small-scale Prism schedule used in our main experiments begins with very narrow head counts ($h_1 = 3$), which admit no valid *grouped* KV configuration: with only 3 query heads a layer permits only $g \in \{1, 3\}$, i.e. MQA or full MHA, with no intermediate grouping. To enable a like-for-like comparison in which every layer is genuinely grouped, both GQA variants use the finest GQA-compatible progressive schedule, $(4, 4, 8, 8, 12, 12, \dots)$ (Config-4 in Table 5), whose per-layer head counts all admit a valid KV-group factorization. For a controlled comparison, the MHA "Prism" row in Table 6 uses this *same* $(4, 4, 8, 8, 12, 12, \dots)$ schedule rather than the headline $h_1 = 3$ configuration, so that the only variable between the MHA and GQA rows is the presence of KV sharing.

Table 6: Composability of the Prism schedule with Grouped-Query Attention at the Small (124M) scale. Both attention families use the same GQA-compatible progressive schedule $(4, 4, 8, 8, 12, 12, \dots)$ (Config-4 in Table 5). Validation metrics are mean $\pm$ standard deviation across three independent seeds. Bold indicates the progressive schedule within each family.

Attention	Head Schedule	n_{params}	Val Loss ↓	Tok/s
MHA (full)	Uniform (12×)	124M	3.3748 ± 0.0027	2875K
	Prism (4-8-12)	124M	$\mathbf{3.3655 \pm 0.0026}$	2875K
GQA	Uniform (12×)	117M	3.3868 ± 0.0046	2921K
	GQA + Prism (4-8-12)	117M	$\mathbf{3.3790 \pm 0.0025}$	2921K

F Implementation Details

F.1 Head Schedule in PyTorch

The Prism Transformer requires only a single structural change to a standard transformer block: the number of attention heads h_l is made *layer-dependent*, drawn from a fixed schedule `n_heads_per_layer`. All weight matrices retain their original $d_{\text{model}} \times d_{\text{model}}$ shape; only the reshape stride in the multi-head split changes per layer.

Algorithm 1 gives the forward pass for a single Prism attention layer. Because the head dimension $d_{\text{head}} = d_{\text{model}}/h_l$ varies across layers, the rotary-embedding buffers $(\cos, \sin)$ are shared via a cache keyed on d_{head}, so each unique head dimension allocates its buffers exactly once regardless of how many layers share that dimension.

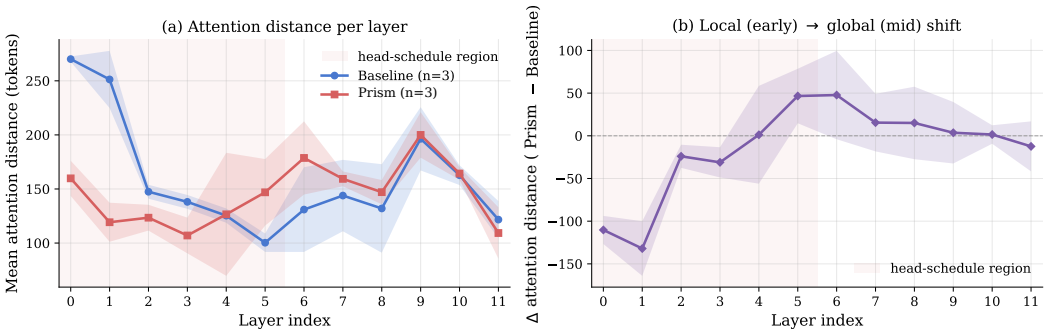
Algorithm 1 Prism Transformer Attention Forward Pass (layer l)

Require: Input $X \in \mathbb{R}^{B \times T \times d_{\text{model}}}$; fused projection $W_{QKV} \in \mathbb{R}^{d_{\text{model}} \times 3d_{\text{model}}}$ (i.e. $[W_Q \mid W_K \mid W_V]$ concatenated column-wise), output projection $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$; head count h_l from the layer schedule; shared rotary cache $\mathcal{R}$
Ensure: Output $Z \in \mathbb{R}^{B \times T \times d_{\text{model}}}$

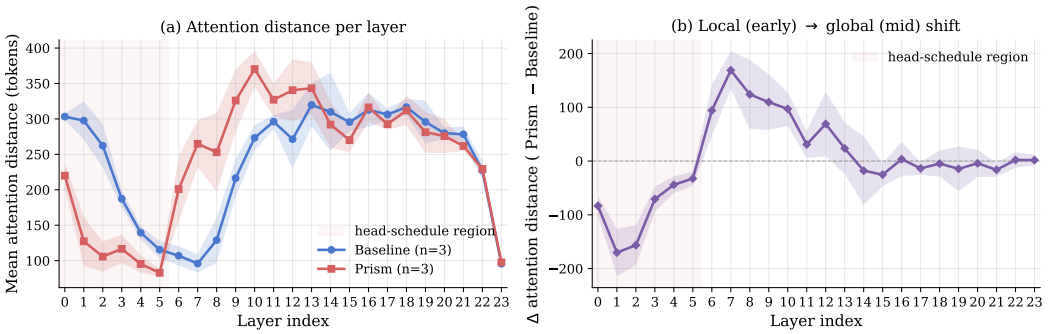
$d_{\text{head}} \leftarrow d_{\text{model}}/h_l$ ▷ per-layer head dimension
$[Q \mid K \mid V] \leftarrow XW_{QKV}$ ▷ single fused linear, shape $(B,\ T,\ 3d_{\text{model}})$
Reshape Q, K, V to $(B,\ T,\ h_l,\ d_{\text{head}})$
rotary $\leftarrow \mathcal{R}[d_{\text{head}}]$ ▷ select cached `Rotary` module for this d_{head}
$(\cos, \sin) \leftarrow \text{rotary}(Q)$ ▷ compute RoPE buffers from sequence length T
$Q \leftarrow \text{RoPE}(Q, \cos, \sin)$; $K \leftarrow \text{RoPE}(K, \cos, \sin)$ ▷ V is not positionally encoded
Transpose Q, K, V to $(B,\ h_l,\ T,\ d_{\text{head}})$ ▷ V unchanged by RoPE
$Y \leftarrow \text{FlashAttn}(Q,\ K,\ V,\ \text{causal} = \top)$ ▷ `scaled_dot_product_attention`
$Y \leftarrow \text{MergeHeads}(Y)$ ▷ $(B, h_l, T, d_{\text{head}}) \rightarrow (B, T, d_{\text{model}})$
$Z \leftarrow YW_O$
return Z

Rotary embeddings: We apply standard rotary positional embeddings (Su et al., 2021) to the queries and keys, with a separate frequency grid per unique head dimension in the schedule, shared

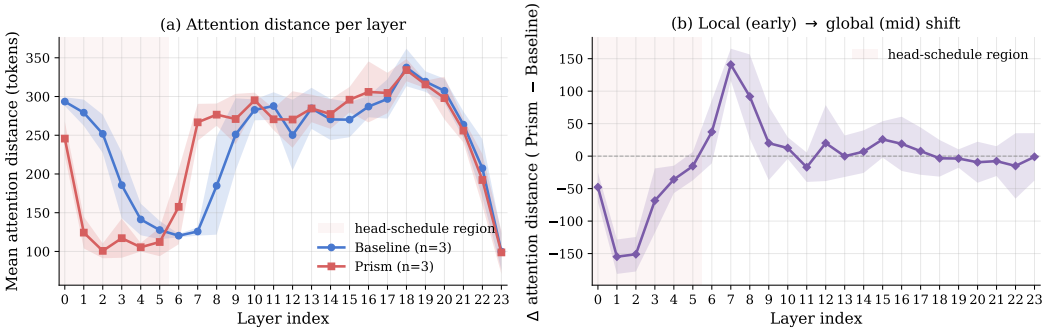
across all layers of that width (the cache $\mathcal{R}$ in Algorithm 1). Keying RoPE to each head's own dimension, rather than sharing one fixed rotary band across the network, keeps the schedule smooth: a single shared band would have to divide every per-layer head dimension, collapsing the admissible head counts to those with a common factor and forcing the schedule into a few large jumps. Per-head-dimension RoPE instead admits the intermediate stages that form the gradual multi-phase staircase.



(a) Small (124M).



(b) Medium (354M).



(c) Large (757M).

Figure 5: **Per-layer attention distance, Prism vs. Baseline, across three scales.** For each scale: *(left)* mean attention distance per layer with ± 1 s.d. bands over $n{=}3$ seeds; *(right)* $\Delta\mathcal{D}_l = \mathcal{D}_l^{\text{Prism}} - \mathcal{D}_l^{\text{Base}}$ with the propagated seed band. The shaded span marks the layers where the two head schedules differ (the progressive phase). Relative to the baseline, Prism attends *more locally* in the early (wide-head) layers and *more globally* in the mid layers where the schedule completes, producing a clear sign change in $\Delta\mathcal{D}_l$. The two models converge in the late layers. The pattern is consistent across all three scales.