GATED RECURRENT TRANSFORMERS: EXPRESSIVE DEPTH THROUGH RECURRENT MODULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling transformer language models creates an inherent tension between expressivity and memory efficiency. While unique weights across layers preserve functional specialization—from input-grounding to abstract refinement—they incur a substantial memory footprint. Conversely, standard depth-sharing enforces uniform transformations that collapse representational diversity and degrade modeling quality. We introduce the GATED RECURRENT TRANSFORMER (GRT), a recurrent depth transformer inspired by gated recurrent neural networks, where a lightweight projection and an elementwise update gate—conditioned on the hidden state, a fixed prelude output, and noise resampled at every step—modulate the recurrent update. This allows the model to specialize the input to the same few layers across recurrences, rather than requiring many unique layers to achieve functional diversity. Under an *iso*FLOPS constraint, a 3-layer GRT matches the accuracy of a 12-layer GPT-2 Small baseline with similar training and inference FLOPs, and leads a wide range of existing recurrent depth approaches in all nine scale-by-budget cells; at medium and large scale it approaches dense quality at the standard token budget and overtakes it at medium scale once that budget is doubled. Under an *iso*PARAMS constraint, deeper recurrence achieves a **2.76** validation loss versus **2.84** for a non-recurrent counterpart at matched parameter and data budget. Our results demonstrate that adaptive depth reuse is a principled strategy for trading parameters for quality. Intermediate recurrences additionally support lossless greedy self-speculative decoding without a separate draft model. Moreover, we propose an optional lightweight post-training optimization, Channel-wise KV Codec, to compress recurrent KV caches to the storage budget of a single K/V pair per token.

1 INTRODUCTION

Scaling transformer language models has delivered remarkable gains, yet it necessitates a simultaneous increase in parameter count and training compute (Kaplan et al., 2020). In standard architectures, depth and parameters are rigidly coupled: every additional layer introduces a fresh set of weights, creating a memory bottleneck that constrains effective depth under fixed hardware budgets.

Beyond memory efficiency, however, there is a deeper computational motivation rooted in the nature of intelligence itself: Turing's foundational insight (Turing, 1936) is that a *finite* set of states and symbols, applied *iteratively*, is sufficient to compute anything computable — suggesting that the power of a reasoning system lies not in its breadth of parameters, but in its depth of iteration. This principle finds a striking modern echo in the test-time compute paradigm (Snell et al., 2025), where models allocate more compute at inference to improve reasoning (Guo et al., 2025; OpenAI, 2024). While existing approaches realize this compute through chain-of-thought steps in the output space—consuming sequence length in the process (Feng et al., 2023)—recurrent depth offers a complementary realization: by iterating a shared transformation over the input representation, the model "thinks" more deeply entirely within its hidden states without generating extra tokens or storing extra parameters.

Weight sharing is the natural response to memory-depth coupling, allowing effective depth to grow without inflating parameter count (Dehghani et al., 2019; Geiping et al., 2025; Lan et al., 2020; Zhu et al., 2025a). Prior work has utilized recurrent depth either as an efficiency tool to match quality at reduced parameter counts (Bae et al., 2025a; Jeddi et al., 2026) or as a performance tool to improve

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.

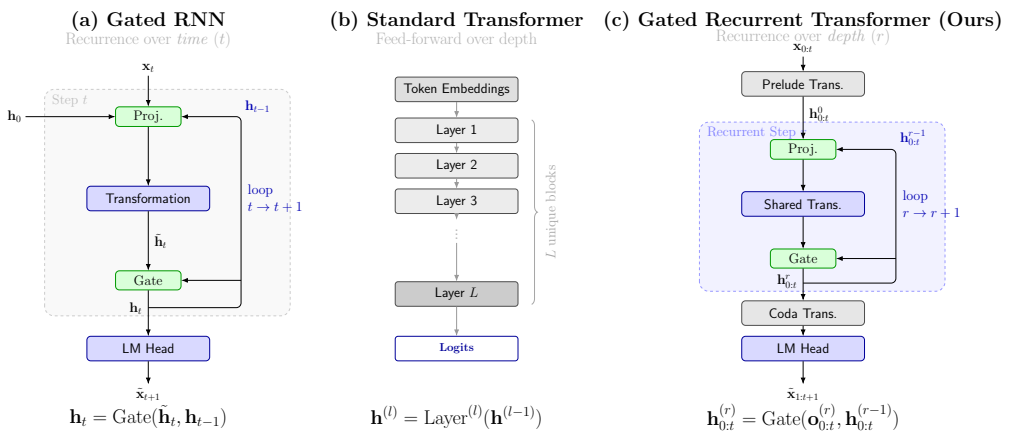


Figure 1: **GRT overview.** (a) A Gated Recurrent Unit (GRU) reuses the same weight matrices at every sequence position—recurrence over the *sequence* dimension. (b) A standard transformer stacks L independent layers, each with unique weights, applied once to the full sequence. (c) GRT introduces recurrence over the *depth* dimension, applying each recurrence to the full sequence

accuracy at matched parameter counts (Geiping et al., 2025; Zhu et al., 2025a). However, a unified treatment of these perspectives is missing, and a deeper architectural tension remains: standard weight sharing forces an identical transformation on an ever-evolving representation. Because the hidden state after the first recurrence differs fundamentally from the eighth, a static transformation collapses the functional diversity that makes depth valuable. This raises the question: Can a single fixed transformation serve meaningfully across such diverse representational stages, or does it inevitably collapse the functional diversity that makes depth valuable in the first place. Such questions motivate the design of GRT.

To resolve this tension, GRT introduces a per-element learned gate that conditions on the current hidden state, a fixed *prelude* representation of the original input (Geiping et al., 2025), and stochastic noise. This constructs a distinct, context-grounded input at every recurrence, allowing a single weight tensor to behave as multiple specialized layers. Gates are initialized such that training starts with the residual stream passing through nearly unchanged through all recurrences. As training progresses, gating emerges gradually as the shared block learns to selectively refine representations. The shared transformation is thus fixed in weights but dynamic in behavior — the model thinks differently at each pass despite reusing the same parameters. Combined with depth sampling during training (Geiping et al., 2025), this architecture enables a continuous compute-quality tradeoff at inference and facilitates evaluation across both *iso*FLOPS and *iso*PARAMS regimes.

We summarize our contributions as follows:

- **Architecture**: We propose GRT, which uses per-element gating and stochastic perturbations to prevent representational collapse, allowing a shared core to behave distinctly across recurrence steps.
- **Unified Evaluation**: We evaluate GRT under both *iso*FLOPS and *iso*PARAMS regimes at three scales. Under *iso*FLOPS it leads MoR and heavy-tail depth sampling in all nine scale-by-budget cells and matches or leads RRT and Ouro at every scale at the standard budget, matches the dense GPT-2 baseline at small scale while using **64%** fewer parameters, and closes on it at medium and large scale as the token budget grows. Under *iso*PARAMS it improves validation loss at matched parameter count, at a higher inference-FLOP cost.
- **Emergent Inference Reuse**: Without auxiliary early-exit training, intermediate recurrences support early exit and lossless greedy self-speculative decoding without a separate draft model.
- **KV Cache Compression**: We introduce the Channel-wise KV Codec, a lightweight post-training compressor that reduces recurrent K/V trajectories to the storage budget of a single K/V pair per token while keeping the trained GRT checkpoint frozen.

2 Related Work

The concept of shared weights across repeated computational steps traces back to the seminal backpropagation paper (Rumelhart et al., 1986), that briefly proposed, at its end, the synchronous iterative net: a network where each iteration corresponds to a layer with tied weights. (Werbos, 1990) extended this to temporal sequence modeling via backpropagation through time (BPTT), inspiring recurrent architectures — RNNs (Elman, 1990), LSTMs (Hochreiter & Schmidhuber, 1997), and GRUs (Cho et al., 2014) — that apply shared weights along the sequence dimension, processing inputs one timestep at a time. In contrast, this work applies recurrent depth across the full input, iterating a shared-weight transformation along network depth to progressively refine the input representation.

In the era of transformers, early recurrent-depth work include ALBERT (Lan et al., 2020), which shared all layer weights in BERT (Devlin et al., 2019), and Universal Transformers (Dehghani et al., 2019), which added per-token adaptive halting. Both, however, shared the *entire* layer stack. Later work instead separated fixed and recurrent stages: (Geiping et al., 2025) introduced prelude, shared-core, and coda layers, while Koishekenov et al. (2025) motivated recurring only the middle "reasoning" layers, identifying earlier layers as "encoding" and later layers as "decoding" layers. Our GRT follows this selective middle-layer recurrence.

Prior work has shown that parameters in feed-forward network (FFN) layers primarily store factual knowledge (Geva et al., 2021), while computational depth drives reasoning ability (Zhu et al., 2025b), motivating recurrence to improve reasoning. We organize prior work on recurrent depth transformers along two evaluation regimes. In the *iso*PARAMS regime — increasing FLOPs by recurring layers while keeping parameter count fixed, trading compute for improved accuracy — (Geiping et al., 2025; Koishekenov et al., 2025; Zhu et al., 2025a) demonstrate consistent quality gains. In the *iso*FLOPS regime — reducing parameter count by recurring layers while matching training and inference compute, trading memory for efficiency — (Jeddi et al., 2026) and (Bae et al., 2025a) show that depth reuse can achieve comparable quality at a smaller memory footprint. Our work spans both regimes, with *iso*FLOPS as our primary contribution and complementary *iso*PARAMS results.

A related line of work dynamically adapts computation across tokens or layers. (Graves, 2017) uses learned per-position halting, (Raposo et al., 2024) routes tokens through subsets of layers, and Mixture-of-Recursions (Bae et al., 2025b) routes tokens to different recursion depths. These methods adapt computation through dedicated halting or routing mechanisms. In contrast, GRT simply modulates how much of the shared-block update is absorbed at each recurrence step, without routing or halting logic.

Speculative decoding (Leviathan et al., 2023) accelerates autoregressive generation with a cheaper draft model whose outputs are verified by the target model. Self-speculative variants remove the draft model: (Zhang et al., 2024) skips intermediate transformer blocks for drafting, while (Elhoushi et al., 2024) exits early. In GRT, an intermediate *recurrence* of the shared core can serve a drafting role, enabling self-speculation without introducing separate early-exit objective or draft model.

KV-cache compression addresses a complementary inference bottleneck. H_2O (Zhang et al., 2023) reduces cache growth by retaining attention heavy hitters, whereas KIVI (Liu et al., 2024) quantizes cached keys and values. Our Channel-wise KV Codec targets a different axis: it retains every token and instead compresses the multiple cache states created by recurrent depth into a single K/V-equivalent latent state per token.

Deep equilibrium models (Bai et al., 2019) use weight-tied iteration toward a fixed point solving for the equilibrium directly and differentiating through it with implicit gradients, avoiding the memory cost of storing an unrolled trajectory. GRT shares the weight-tied update but not the equilibrium objective: we unroll a fixed number of discrete steps and backpropagate through them, and depth sampling trains each step to serve as an exit rather than as an approach to a single limit.

Scaling laws (Hoffmann et al., 2022; Kaplan et al., 2020) have established how model size and data jointly determine performance. Notably, Kaplan et al. (2020) directly evaluated scaling laws of parameter-sharing transformers and observed that recurrent models perform better at matched parameter count (*iso*PARAMS) but worse at matched compute (*iso*FLOPS). In our GRT, we propose a novel recurrent depth architecture that aims to improve the scaling laws of loss versus parameter count and loss versus training FLOPs, demonstrating consistent gains under both *iso*FLOPS and *iso*PARAMS regimes as model size and training data scale.

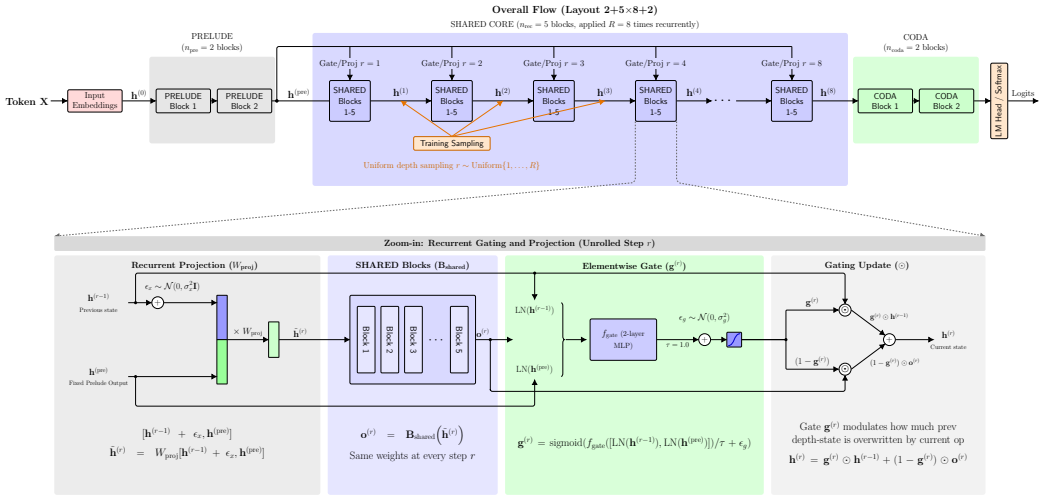


Figure 2: **GRT architecture (2+5×8+2 layout).** *Top:* The prelude produces a fixed context $\mathbf{h}^{(\text{pre})}$, while the shared core recurrently refines $\mathbf{h}^{(r)}$ before the coda. *Bottom:* Zoom-in of a single recurrence step r: the current state $\mathbf{h}^{(r-1)}$ is projected together with $\mathbf{h}^{(\text{pre})}$ via W_{proj}, processed by the shared blocks to produce proposal $\mathbf{o}^{(r)}$, and blended back via the elementwise gate $\mathbf{g}^{(r)} \in [0,1]^{T \times d}$.

3 Recurrent Depth Reuse for Language Modeling

Preliminaries Let $\mathbf{X} = (x_1, \ldots, x_T)$ be a token sequence of length T drawn from vocabulary $\mathcal{V}$. A standard autoregressive transformer with L blocks defines a chain of residual updates: $\mathbf{h}^{(\ell)} = \mathbf{h}^{(\ell-1)} + \text{Block}_\ell(\mathbf{h}^{(\ell-1)})$ for $\ell = 1, \ldots, L$, where d is the embedding dimension and $\mathbf{h}^{(0)} \in \mathbb{R}^{T \times d}$ are initial token embeddings. Each block comprises pre-norm multi-head self-attention and a token-wise MLP. The objective is to minimize the negative log-likelihood, $\mathcal{L} = -\sum_t \log p(x_t \mid x_{<t})$. Standard transformers couple depth and parameters rigidly, as every additional layer adds a fresh set of weights. The total unique parameter count scales as $\Theta(L \cdot d^2)$.

The Prelude–Shared–Coda Architecture Following (Geiping et al., 2025), GRT partitions transformer blocks as $n_{\text{pre}}+n_{\text{rec}} \times R+n_{\text{coda}}$, separating fixed prelude and coda blocks from a recurrent shared core:

- **Prelude (n_{pre} blocks):** Applied once to produce a fixed conditioning signal $\mathbf{h}^{(\text{pre})} = \text{Block}_{n_{\text{pre}}} \circ \cdots \circ \text{Block}_1(\mathbf{h}^{(0)})$.
- **Shared core (n_{rec} blocks):** Applied recurrently R times, processing the evolving state $\mathbf{h}^{(r-1)}$ conditioned on the fixed prelude anchor $\mathbf{h}^{(\text{pre})}$ and stochastic noise ϵ_x: $\mathbf{h}^{(R)} = \text{Block}_{n_{\text{shared}}} \circ \cdots \circ \text{Block}_1[\epsilon_x, \mathbf{h}^{(\text{pre})}]$. The recurrent update is detailed in the next section.
- **Coda (n_{coda} blocks):** Maps the final recurrent state to $\mathbf{h}^{(\text{coda})} = \text{Block}_{n_{\text{coda}}} \circ \cdots \circ \text{Block}_1(\mathbf{h}^{(R)})$, which is projected into the language modeling heads to produce logits.

The unique parameter count is $\Theta\big((n_{\text{pre}}+n_{\text{rec}}+n_{\text{coda}})d^2\big)$, independent of R. For example, $2{+}5{\times}4{+}2$ executes $2 + 5 \cdot 4 + 2 = 24$ blocks while storing only $2 + 5 + 2 = 9$ ($2.6\times$ fewer) unique blocks. Figure 2 illustrates the architecture. Forward-pass FLOP accounting is provided in Appendix A.2.

Recurrent Projection and Elementwise Gate Merely feeding the output of each recurrence as the input to the next recurrence risks forgetting the original input context, suffers from vanishing or exploding gradients across deep recurrences, and taxes the representational capacity of a single shared weight tensor.

Inspired by RNNs (Elman, 1990) in general, and GRUs (Cho et al., 2014) in particular, we treat the hidden representation after each recurrence step as a *state* that evolves across recurrences (as illustrated in Figure 1). Each application of the shared core reads this state, transforms it, and writes an updated state back—analogous to an RNN cell, but operating over depth rather than time. This

framing motivates the gating mechanism below: just as GRU gates control how much of its cell state is overwritten at each timestep, GRT's elementwise gate controls how much of the current depth-state is replaced by the shared block's output.

Hence, we introduce our novel adaptive gating approach that aims to mitigate the 3 aforementioned problems. At each recurrence step r, GRT prepares the input to $\mathcal{B}_{\text{shared}}$ by projecting the current state together with the fixed prelude output, then selectively writes the block's output back to the residual stream via a learned elementwise gate. The full update is:

$$\tilde{\mathbf{h}}^{(r)} = W_{\text{proj}}\left[\mathbf{h}^{(r-1)} + \epsilon_x,\ \mathbf{h}^{(\text{pre})}\right], \tag{1}$$

$$\mathbf{o}^{(r)} = \mathcal{B}_{\text{shared}}\left(\tilde{\mathbf{h}}^{(r)}\right), \tag{2}$$

$$\mathbf{h}^{(r)} = \mathbf{g}^{(r)} \odot \mathbf{h}^{(r-1)} + \left(1 - \mathbf{g}^{(r)}\right) \odot \mathbf{o}^{(r)}, \tag{3}$$

where $[\cdot, \cdot]$ denotes concatenation along the feature axis, $\mathbf{g}^{(r)} \in [0, 1]^{T \times d}$ is a learned elementwise gate (defined in full below in Eq. (4)), $W_{\text{proj}} \in \mathbb{R}^{d \times 2d}$ is a learned projection, $\epsilon_x \sim \mathcal{N}(0,\, \sigma^2 \mathbf{I})$ is injected additive noise, and $\odot$ is elementwise multiplication.

The gate $\mathbf{g}^{(r)} \in [0, 1]^{T \times d}$ is produced by a small feed-forward network conditioned on the normalised current state and the normalised prelude output:

$$\mathbf{g}^{(r)} = \sigma\Big(f_{\mathbf{g}}\big([\text{LN}(\mathbf{h}^{(r-1)}),\ \text{LN}(\mathbf{h}^{(\text{pre})})]\big)/\tau + \epsilon_g\Big), \tag{4}$$

where $\sigma(\cdot) = 1/(1 + e^{-(\cdot)})$ is the sigmoid function, LN denotes layer normalization (Ba et al., 2016), τ is a temperature hyperparameter (set to 1.0 throughout our experiments), $\epsilon_g \sim \mathcal{N}(0,\, \sigma_g^2)$ is per-scalar gate noise injected during training, and $f_{\mathbf{g}}$ is a two-layer MLP with SiLU activation and hidden dimension $d_{\text{gate}} = d$. The second linear layer of $f_{\mathbf{g}}$ has its bias initialised to $+4$, placing $\mathbf{g}^{(r)} \approx 0.98$ at the start of training: the copy branch of Eq. (3) dominates, the residual stream passes through the recurrence almost unchanged, and the model learns which elements to overwrite as training proceeds. Positive initialisation of a forget gate is long-standing practice in recurrent networks (Gers et al., 2000), and Jozefowicz et al. (2015) found a bias of $+1$ or $+2$ enough to bring a vanilla LSTM level with the best variants in their search. We simply apply the same principle over depth rather than time. We swept across $\{-2, 0, +2, +4\}$ at the full training horizon the spread is 0.019 nats (Section D.6).

By defining $\mathbf{g}_{t,i}^{(r)}$ as the i^{th} element of the gate for token t at recurrence r, GRT achieves a highly granular, per-element specialization. When $\mathbf{g}_{t,i}^{(r)} \to \mathbf{1}$, the residual stream remains unchanged, whereas $\mathbf{g}_{t,i}^{(r)} \to \mathbf{0}$ allows the block output to fully replace the state. This per-recurrence specialization provides a more memory-efficient alternative to approaches such as Per-Layer Embeddings (PLE) (used in recent models like Gemma3n (Google DeepMind, 2025) and Gemma-4 (Google DeepMind, 2026)). While PLE requires L unique embedding layers to provide layer-specific context, our approach conditions the gate on $\mathbf{h}^{(\text{pre})}$ to provide a constant view of the *original* input at every step, while modulating it with $\mathbf{h}^{r-1}$ and stochasticity ϵ_g.

State noise ϵ_x (Eq. (1)) perturbs the recurrent projection input, discouraging the model from learning brittle exact-match patterns across steps. Gate noise ϵ_g (Eq. (4)) prevents the gate from collapsing to a near-constant value during training. Specific noise magnitudes (σ_x, σ_g) are reported in Section A.1.

Recurrence Depth Sampling During Training At each training step, we sample $r \sim \text{Uniform}\{1, \ldots, R\}$. This trains all intermediate exits, enabling effective early exit at inference without auxiliary losses (Chen et al., 2024; Elhoushi et al., 2024), while providing stochastic-depth regularization that improves over fixed-depth training (as shown later in our ablations Section 5.

We summarize the forward pass of GRT in Listing 1.

Channel-wise KV Codec Recurrent depth reduces weight memory but not KV-cache memory: a shared layer that runs R times has one set of weights but produces R recurrent cache states $\{(K^{(r)}, V^{(r)})\}_{r=1}^{R}$. This can erode the memory savings of weight sharing for long-context and

agentic workloads, where KV-cache memory can be substantial. To address this, we introduce an optional post-training technique: Channel-wise KV Codec. For each new token, an encoder E compresses its recurrent K/V states at $r = 1$ and $R - 1$ into a K/V-equivalent latent state $Z = E(K^{(1)}, V^{(1)}, K^{(R-1)}, V^{(R-1)})$, which is then stored in the cache. For subsequent tokens, a recurrence-specific decoder reconstructs the required historical cache as $(\hat{K}^{(r)}, \hat{V}^{(r)}) = D_r(Z)$. We train E and $\{D_r\}_{r=1}^{R}$ with all GRT parameters frozen, first to reconstruct the original recurrent K/V states and then to minimize KL divergence from the full-cache model. This reduces recurrent KV storage from R K/V states to one per token. Full details are in Appendix B.

4 Experiments & Results

Experimental Setup All models are trained from scratch for 20,000 steps (≈9.8B tokens) under a shared nanoGPT (Karpathy, 2022) recipe. We compare against dense GPT-2 baselines and four recurrent competitors—MoR (Bae et al., 2025b), heavy-tail Poisson depth sampling (Geiping et al., 2025), RRT (Bae et al., 2025a), and Ouro (Zhu et al., 2025a)—trained under the same setup. Full optimization, architecture, and hardware details are provided in Appendix A.1. *iso*FLOPs means matching the number of Transformer-block applications, which dominate compute.

Main Results Table 1 summarizes performance across model sizes. In the ***iso*FLOPs regime**, GRT achieves competitive quality while using only **36–37%** of the dense baseline's parameters. At small scale, it matches the dense baseline and is robust across seeds (3.145 ± 0.004; Appendix D.1). At medium and large scale, the dense baseline leads by only 0.05 and 0.06 nats, respectively. In Section 5, we show both gaps narrow when we train on more tokens. Comparing with other recurrent depth methods, GRT achieves the best validation loss across all model scales, with its advantage over such methods generally increasing with scale. Note that while some approaches, such as MoR (Bae et al., 2025b) and RRT (Bae et al., 2025a), introduce recurrence-specific parameters, GRT retains strict weight sharing while using input-dependent gating to specialize each recurrence.

In the ***iso*Params regime**, recurrence improves quality at a fixed parameter budget, reducing loss by **0.08** nats at medium scale and **0.06** at large scale. Thus, recurrent depth provides a direct tradeoff between inference compute and model quality without increasing the parameter footprint.

Table 2 reports accuracy across 9 benchmarks at large scale using `lm-eval-harness` (Gao et al., 2024). The *iso*FLOPs variant matches the dense model on average (42.08 vs. 42.05), confirming that parameter efficiency translates across evaluation protocols, while the *iso*Params variant outperforms the dense baseline on 8 of 9 tasks, consistent with the validation-loss advantage in Table 1.

Early Exit Analysis As shown in Figs. 3a and 3b, GRT exhibits emergent early-exit behavior across model scales: intermediate recurrences provide a continuous "compute–quality dial," enabled by uniform depth sampling during training (Section 3), allowing a single checkpoint to trade inference compute for quality without retraining or auxiliary losses. GRT leads all baselines from k=3 until the dense model catches up at full depth. Figure 4 illustrates how predictions evolve across recurrences: knowledge-intensive predictions often stabilize early, while reasoning and open-ended prompts continue to sharpen or refine with depth, suggesting that recurrent computation provides additional "thinking time" when useful.

Self-Speculative Decoding These intermediate exits also enable self-speculative decoding: recurrence $r_d < R$ drafts tokens while the full-R path verifies them, without a separate draft model or auxiliary training. Across ten task settings, this yields $1.18\times$–$1.46\times$ wall-clock speedups with lossless greedy verification; full results are in Appendix C.1.

KV cache Compression Naively, GRT requires a distinct recurrent KV cache at each recurrence step. Following (Zhu et al., 2025a), we first evaluate three training-free sharing rules that retain one recurrent K/V state per token: the first-step cache, the last-step cache, or the recurrence-wise average. Averaging is the strongest baseline on the large *iso*FLOPs checkpoint, reaching 33.4% HellaSwag compared with 35.7% for the full six-cache model (Figure 7).

The K/V analysis in Appendix G.11 shows that recurrent cache states are strongly structured: K changes little across recurrence, whereas V follows a larger early-to-late transition. Motivated

Table 1: **Main results (validation loss).** *Upper*: *iso*FLOPs regime; *lower*: *iso*PARAMS regime. **Layers** denotes unique transformer blocks. ↓ lower is better.

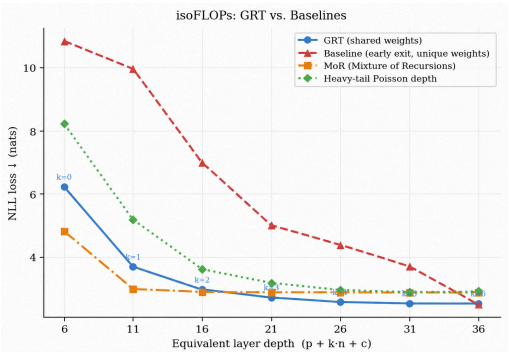
Model	Config	Layers	FLOPs/fwd	Params	Val Loss ↓
*iso*FLOPs *regime*					
GPT-2 Small					
Baseline	12L	12	0.208	124M	3.15
MoR (Bae et al., 2025b)	1+1×10+1	3	0.221	45M	3.30
Heavy-tail Poisson (Geiping et al., 2025)	1+1×10+1	3	0.231	34M	3.23
Ouro (Zhu et al., 2025a)	1+1×10+1	3	0.231	35M	3.19
RRT (Bae et al., 2025a)	1+1×10+1	3	0.237	33M	3.14
GRT (ours)	1+1×10+1	3	0.267	35M	**3.14**
GPT-2 Medium					
Baseline	24L	24	0.705	354M	2.84
MoR (Bae et al., 2025b)	2+5×4+2	9	0.710	137M	3.02
Heavy-tail Poisson (Geiping et al., 2025)	2+5×4+2	9	0.721	124M	2.97
Ouro (Zhu et al., 2025a)	2+5×4+2	9	0.721	126M	2.93
RRT (Bae et al., 2025a)	2+5×4+2	9	0.737	125M	2.95
GRT (ours)	2+5×4+2	9	0.747	127M	**2.89**
GPT-2 Large					
Baseline	36L	36	1.604	774M	2.71
MoR (Bae et al., 2025b)	1+5×6+5	11	1.624	309M	2.91
Heavy-tail Poisson (Geiping et al., 2025)	1+5×6+5	11	1.644	291M	2.93
Ouro (Zhu et al., 2025a)	1+5×6+5	11	1.644	290M	2.82
RRT (Bae et al., 2025a)	1+5×6+5	11	1.673	289M	2.85
GRT (ours)	1+5×6+5	11	1.703	293M	**2.77**
*iso*PARAMS *regime*					
GPT-2 Small					
Baseline	12L	12	0.208	124M	3.15
MoR (Bae et al., 2025b)	1+10×10+1	12	1.679	135M	3.12
Heavy-tail Poisson (Geiping et al., 2025)	1+10×10+1	12	1.788	124M	3.06
Ouro (Zhu et al., 2025a)	1+10×10+1	12	1.788	127M	3.10
RRT (Bae et al., 2025a)	1+10×10+1	12	1.847	125M	3.08
GRT (ours)	1+10×10+1	12	1.824	127M	**3.04**
GPT-2 Medium					
Baseline	24L	24	0.705	354M	2.84
MoR (Bae et al., 2025b)	2+20×4+2	24	2.333	367M	2.78
Heavy-tail Poisson (Geiping et al., 2025)	2+20×4+2	24	2.483	356M	**2.74**
GRT (ours)	2+20×4+2	24	2.508	357M	2.76
GPT-2 Large					
Baseline	36L	36	1.604	774M	2.71
MoR (Bae et al., 2025b)	3+30×6+3	36	8.094	795M	2.69
Heavy-tail Poisson (Geiping et al., 2025)	3+30×6+3	36	8.328	776M	2.70
GRT (ours)	3+30×6+3	36	8.387	779M	**2.65**

Table 2: **Downstream task evaluation (large scale).** Zero-shot accuracy (%; ↑) on standard benchmarks. Superscripts denote deltas relative to GPT-2 Large. **Orange**: best *iso*FLOPs result; **blue**: best *iso*PARAMS result. HSwag denotes HellaSwag; LAMB-OAI/Std denote the OpenAI/standard LAMBADA variants; WinoG denotes WinoGrande. The *iso*PARAMS model improves the average by **+2.10 points**; the *iso*FLOPs model matches GPT-2 Large on average at 37% of its parameters.

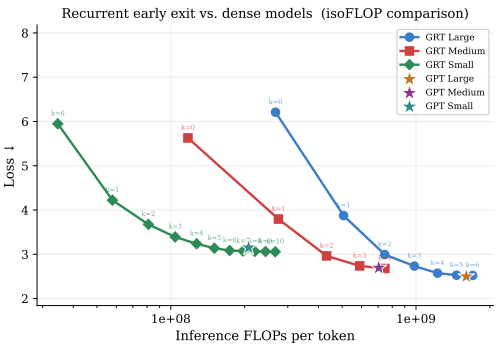
Model	ARC-C	ARC-E	BoolQ	HSwag	LAMB-OAI	LAMB-Std	OBQA	PIQA	WinoG	Avg.
GRT *iso*FLOPs	**25.43**	42.89	**60.18**	35.62	39.05	30.60	28.40	64.31	52.25	**42.08**
GPT-2 Large	23.63	43.31	57.58	37.21	40.05	29.73	29.80	65.51	51.62	42.05
GRT *iso*PARAMS	25.26	**45.54**	52.29	**41.52**	**45.27**	**38.02**	**30.20**	**66.05**	**53.28**	**44.15**

Configs: GRT *iso*FLOPs = 1+5×6+5, 288M; GPT-2 Large = 36L, 774M; GRT *iso*PARAMS = 3+30×6+3, 779M.

by this structure, we apply our proposed Channel-wise KV Codec, which compresses recurrent K/V trajectories to the same one-state budget while keeping the trained GRT checkpoint frozen. On the large model, it reaches **34.8%** HellaSwag, compared with 35.7% for the full cache and 33.4% for recurrence-wise averaging (Figure 7). Post-training details and results across scales are in Appendix B; wall-clock latency and decoding memory are in Appendix C.2..



(a) Early-exit Validation loss vs. equivalent layer depth $p + k \cdot n + c$.



(b) Early-exit Validation loss vs. inference FLOPs per token at varying recurrence depths k.

Figure 3: **Early-Exit Analysis** Inference FLOPs using sequence length 1024, batch size 1.

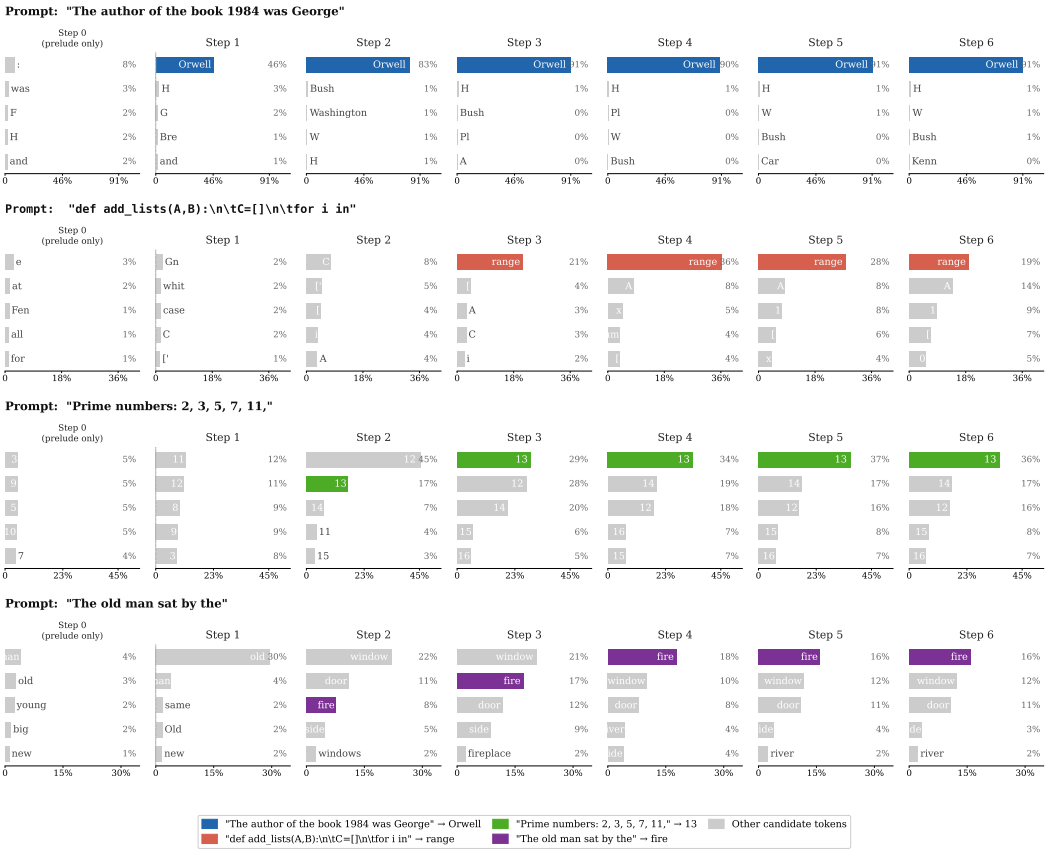


Figure 4: **Next-token probability at each recurrence step.** Each row shows one prompt; the correct answer is highlighted in colour. Prelude-only output (leftmost column) captures partial signal; by step 2 the correct token dominates in three of four prompts.

Mechanistic Analysis Summary Appendix G analyzes the large 1+5×6+5 checkpoint in detail, and we provide a summary here. Figure 11 shows that the gate, $g^{(r)}$, evolves from write-heavy early recurrences to copy-heavy later ones, consistent with coarse-to-fine refinement. Figure 13 shows that recurrent representations remain more similar than layers of a matched-depth dense transformer, consistent with the shared block learning a reusable rather than depth-specialized transformation. Finally, Figure 14 shows that harder tokens accumulate $10\times$ more improvement across recurrences, indicating implicit compute routing without an explicit difficulty signal.

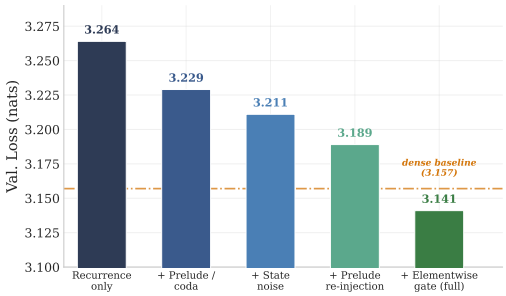


Figure 5: **Component ablation on GRT-small (20,000 steps).** Each bar adds one component to the one on its left. The dashed line is the dense 12-layer baseline at 3.157, trained identically; only the full model falls below it. The 2,000-step counterpart on the medium configuration is in Appendix D.2.

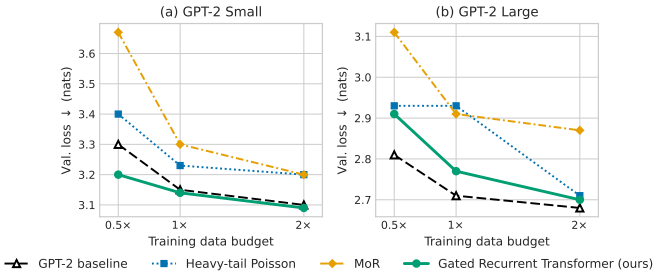


Figure 6: **Validation loss vs. training data budget.** Validation loss at small and large scale as the training budget varies.

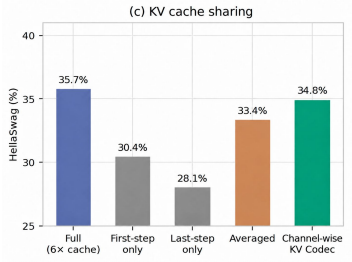


Figure 7: **KV cache compression.** HellaSwag accuracy (%) on the Large *iso*FLOPS checkpoint.

5 Ablations

Component Ablation Figure 5 traces a sequential ablation on the small model at the full 20,000-step budget. Recurrence alone raises validation loss by 0.107 nats: without a mechanism to differentiate steps, the shared block conflates early and late processing into a single weight tensor. Adding prelude and coda boundaries (row 3) recovers 0.035 nats; state noise (row 4) contributes 0.018 more, which also isolates the value of injecting noise at every step rather than only at initialisation. Prelude re-injection into each recurrence (row 5) cuts loss by a further 0.022 nats. The elementwise gate (row 6) is the single largest contributor at $-$**0.048 nats**.

Data Efficiency As illustrated in Figure 6, we analyze the performance of recurrent depth across varying training data budgets. At smaller model scales, GRT consistently maintains a superior Pareto frontier compared to the dense baseline across all observed data volumes. At larger scales, a more complex "crossover" dynamic emerges: while the dense model shows an initial lead under low-data regimes, the gap narrows significantly as the token budget increases. For the Large-scale configuration, the trajectories converge at higher data budgets, suggesting that recurrent sharing is particularly effective at saturating parameter capacity when provided with sufficient data.

6 Conclusion

GRT decouples model depth from parameter count through gated recurrent depth reuse. Under *iso*FLOPS, it outperforms other recurrent-depth approaches across scales while using only 36–37% of the dense baseline's unique parameters. At small scale, GRT also outperforms the dense baseline; at larger scales, a modest gap remains but narrows as the training-token budget increases, suggesting recurrent sharing benefits from the larger data budgets typical of modern LLM training. Under *iso*PARAMS, additional recurrent depth further improves validation loss and downstream performance by up to $+8.29$ points. To further reduce inference memory, we introduce a lightweight post-training KV-cache compression method that compresses the recurrent KV history to one K/V-equivalent state per token while keeping the base model frozen. Limitations include fixed recurrence depth, potential hyperparameter re-tuning beyond GPT-2, and a scale-dependent optimal sharing fraction. Future work includes adaptive per-token depth and distillation from dense models.

AI USE STATEMENT

Generative AI tools were used to draft paragraphs in this paper and to help polish the writing, improve the clarity of the manuscript, and shorten it to abide by the page limit. Generated text was verified and edited by the authors. Generative AI tools were also used to generate code that was used to run the experiments, but ideas, analyses, and conclusions were produced by the authors.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise loRA. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=WwpYSOkkCt.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=QuqsEIVWIG.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. In *The Forty-first International Conference on Machine Learning*, 2024.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

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

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: https://doi.org/10.1207/s15516709cog1402_1. URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=qHrADgAdYu.

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, 07 2024. URL https://zenodo.org/records/12608602.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. URL https://arxiv.org/abs/2502.05171.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446/.

Google DeepMind. Gemma 3n model overview. https://ai.google.dev/gemma/docs/gemma-3n, May 2025. Introduced Per-Layer Embeddings (PLE) for on-device LLMs.

Google DeepMind. Gemma 4 model overview. https://ai.google.dev/gemma/docs/core, April 2026. Accessed: 2026-05-06. Last updated 2026-04-16.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

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, 2022. URL https://arxiv.org/abs/2203.15556.

Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5(2):248–264, 1975. doi: 10.1016/0047-259X(75)90042-1.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2342–2350, 2015.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts, 2025. URL https://arxiv.org/abs/2510.07358.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286, 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024.

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

OpenAI. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, September 2024.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. doi: 10.1038/323533a0.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936. doi: 10.2307/2268810.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2025a. URL https://arxiv.org/abs/2510.25741.

Ruike Zhu, Hanwen Zhang, Kevin Li, Tianyu Shi, Yiqun Duan, Chi Wang, Tianyi Zhou, Arindam Banerjee, and Zengyi Qin. Beyond parameters: Exploring virtual logic depth for scaling laws, 2025b. URL https://arxiv.org/abs/2506.18233.

A Implementation Details

A.1 Experimental Setup

We build on the nanoGPT codebase (Karpathy, 2022) for our transformer backbone implementation. All models are trained with sequence length $T = 1024$ tokens and GPT-2 BPE tokenisation (50,257-token vocabulary). We use AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, a 2,000-step linear learning-rate warmup, and a cosine learning-rate schedule from a peak of 6×10^{-4} to 6×10^{-5}. Gradient norms are clipped at 1.0, and all runs use `bfloat16` mixed precision. State noise $\sigma_x = 0.1$ and gate noise $\sigma_g = 0.1$ are used throughout all GRT runs; the gate temperature is fixed at $\tau = 1.0$.

We train for 20,000 steps with an effective batch size of approximately 491,520 tokens per step (batch size 8, gradient accumulation 60, sequence length 1024), totalling approximately **9.8B** tokens per run. Dense baselines (GPT-2 small, medium, and large (Radford et al., 2019)) are trained from scratch under the identical setup; no pre-trained weights are used. We train four recurrent competitors from scratch under the same recipe: *Mixture-of-Recursions* (MoR) (Bae et al., 2025b); a variant following Geiping et al. (2025) that uses the prelude–coda layout with heavy-tail Poisson depth sampling; *Relaxed Recursive Transformers* (RRT) (Bae et al., 2025a), which relaxes weight tying with per-recurrence LoRA adapters; and *Ouro* (Zhu et al., 2025a), which supervises every loop iteration. All reported runs use 2× NVIDIA H200 GPUs; the complete per-run configurations are listed below.

```
def grt_forward(x, model, R, r_min=1):        # max depth R
    h = token_embed(x) + pos_embed(x)              # [S, d]
    for block in model.prelude_blocks:
        h = h + block(h)
    h_pre = h                                      # frozen anchor
    r = R if not training else uniform_sample(r_min, R)
                                                   # sample depth in training
    for step in range(r):                    # r = 0,...,R-1
        eps = sample_gaussian(0, sigma**2)
        h_tilde = W_proj(cat([h + eps, h_pre]))     # re-entry projection
        o = B_shared(h_tilde)                       # shared weights
        eps_g = sample_gaussian(0, sigma_g**2)
        g = sigmoid(f_gate(LN(h), LN(h_pre))/tau + eps_g)
                                                   # elementwise gate
        h = g * h + (1 - g) * o                    # gated update
    for block in model.coda_blocks:
        h = h + block(h)
    return lm_head(h)                           # [S, |V|]
```

Listing 1: **GRT forward pass.**

A.2 FLOP Accounting

Per-token forward-pass cost for a n_{pre}+$n_{\text{rec}} \times R$+n_{coda} configuration at sequence length S is

$$\text{FLOPs} = \big(n_{\text{pre}} + n_{\text{rec}} R + n_{\text{coda}}\big)\big(24d^2 + 4Sd\big) + R \cdot 10d^2, \tag{5}$$

where $24d^2 + 4Sd$ is the standard per-block cost (attention and MLP projections plus the two attention matmuls) and $10d^2$ is the per-step overhead of the recurrent projection W_{proj} and gate MLP $f_{\mathbf{g}}$.

A.3 Hyperparameter Table

Table 3 lists the full training configuration for every run reported in the main paper.

All recurrent runs use gate bias initialisation $+4$ ($\mathbf{g} \approx 0.98$ at init), state noise $\sigma_x = 0.1$, gate noise $\sigma_g = 0.1$, and gate temperature $\tau = 1.0$. During training, recurrence depth is sampled uniformly from $\{1, \ldots, R\}$ at each training step.

Table 3: **Training configurations.** p: prelude blocks; b: shared blocks; R: recurrence steps; c: coda blocks; d: embedding dimension; h: attention heads. Batch tokens = batch_size $\times$ gradient_accumulation $\times$ sequence_length $\times$ num_GPUs. All runs use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$), weight decay 0.1, gradient clip 1.0, bfloat16 mixed precision, cosine LR schedule.

Run	p	b	R	c	d	h	Batch tokens	Peak LR	Steps	Hardware
GPT-2 Small (baseline)	—	—	—	—	768	12	491,520	6×10^{-4}	20k	2× H200
GRT Small (isoFLOP)	1	1	10	1	768	12	491,520	6×10^{-4}	20k	2× H200
GPT-2 Medium (baseline)	—	—	—	—	1024	16	491,520	6×10^{-4}	20k	2× H200
GRT Medium (isoFLOP)	2	5	4	2	1024	16	491,520	6×10^{-4}	20k	2× H200
GRT Medium (isoParam)	2	20	4	2	1024	16	491,520	6×10^{-4}	20k	2× H200
GPT-2 Large (baseline)	—	—	—	—	1280	20	491,520	6×10^{-4}	20k	2× H200
GRT Large (isoFLOP)	1	5	6	5	1280	20	491,520	6×10^{-4}	20k	2× H200
GRT Large (isoParam)	3	30	6	3	1280	20	491,520	6×10^{-4}	20k	2× H200

B Channel-wise KV Codec

The Channel-wise KV Codec is a post-training compression module for the recurrent K/V cache; all GRT parameters remain frozen. Let l index a recurrent physical block, h an attention head, and $j \in \{1, \ldots, d_h\}$ a feature channel within that head, where d_h is the head dimension. For token t, we retain the cache coordinates from recurrences $a = 1$ and $b = R - 1$:

$$\mathbf{x}_{t,l,h,j} = \begin{bmatrix} k^{(a)}_{t,l,h,j} & v^{(a)}_{t,l,h,j} & k^{(b)}_{t,l,h,j} & v^{(b)}_{t,l,h,j} \end{bmatrix}^\top \in \mathbb{R}^4. \tag{6}$$

After centering by the calibration mean $\boldsymbol{\mu}^x_{l,h,j} \in \mathbb{R}^4$, each channel is encoded into two latent scalars:

$$\mathbf{z}_{t,l,h,j} = E^\top_{l,h,j} \left(\mathbf{x}_{t,l,h,j} - \boldsymbol{\mu}^x_{l,h,j} \right), \qquad E_{l,h,j} \in \mathbb{R}^{4\times 2}. \tag{7}$$

The resulting $2d_h$ stored scalars per head match the storage of a single K/V pair. At recurrence r, the corresponding cache coordinates are reconstructed as

$$\begin{bmatrix} \widehat{k}^{(r)}_{t,l,h,j} \\ \widehat{v}^{(r)}_{t,l,h,j} \end{bmatrix} = D^{(r)\top}_{l,h,j} \mathbf{z}_{t,l,h,j} + \boldsymbol{\mu}^{y,(r)}_{l,h,j}, \qquad D^{(r)}_{l,h,j} \in \mathbb{R}^{2\times 2}, \tag{8}$$

where $\boldsymbol{\mu}^{y,(r)}_{l,h,j} \in \mathbb{R}^2$ is the calibration mean of $\begin{bmatrix} k^{(r)}_{t,l,h,j}, v^{(r)}_{t,l,h,j} \end{bmatrix}^\top$.

We initialise each channel with rank-2 reduced-rank regression (RRR) (Izenman, 1975) using ridge coefficient $\lambda = 10^{-4}$, then orthonormalise the resulting two-dimensional latent basis. Initialisation uses 8,192 calibration tokens, with each per-channel linear fit capped at 4,096 samples. The calibration means remain fixed, and only the encoders $E_{l,h,j}$ and recurrence-specific decoders $D^{(r)}_{l,h,j}$ are subsequently optimised.

The first post-training stage runs 400 AdamW steps at learning rate 3×10^{-4} and minimises

$$\mathcal{L}_{\text{rec}} = 0.15\,\text{NMSE}(K) + \text{NMSE}(V) + 10^{-4}\mathcal{L}_{\text{orth}}, \tag{9}$$

where $\mathcal{L}_{\text{orth}}$ penalises deviations of each 4×2 encoder from orthonormal columns. We then minimise KL divergence to the full-cache GRT model during autoregressive decoding. This stage uses learning rate 10^{-4}, KL temperature 1 and orthogonality weight 5×10^{-4}. We use the same optimisation schedule at all three model scales; the codec dimensions and recurrence count follow the corresponding GRT checkpoint.

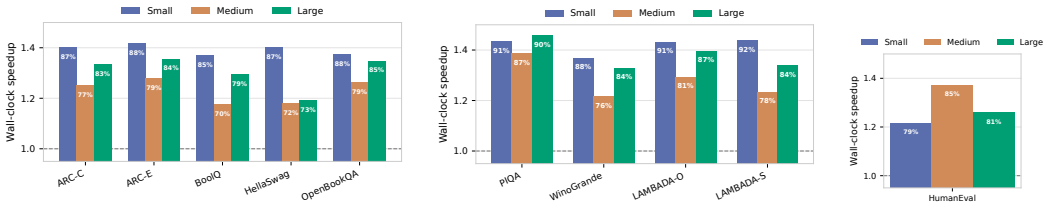
C Additional Inference Results

C.1 Self-Speculative Decoding

Intermediate recurrence exits also provide a natural self-drafter. We use self-speculative decoding (Elhoushi et al., 2024; Leviathan et al., 2023; Zhang et al., 2024), taking recurrence $r_d < R$ as the

Table 4: **Channel-wise KV Codec across model scales.** The base GRT checkpoint is frozen in every row. "Recurrent" is the cache reduction within shared recurrent blocks; "total" includes the unchanged prelude and coda caches. KL, ΔNLL, and top-1 agreement are measured against the corresponding full-cache model.

Scale	Codec params	Recurrent	Total	KL ↓	ΔNLL ↓	Top-1 ↑
Small	36.9K	10.0×	4.00×	0.0319	+0.0201	90.8%
Medium	122.9K	4.0×	2.67×	0.0356	+0.0368	88.4%
Large	204.8K	6.0×	3.27×	0.0493	+0.0487	88.2%



(a) Reasoning and question answering. (b) Completion and commonsense. (c) HumanEval.

Figure 8: **Task-level self-speculative decoding.** Wall-clock speedup over full-depth autoregressive decoding for Small, Medium, and Large checkpoints. Percentages inside bars denote draft-token acceptance. Draft depth and proposal length are fixed per model across all tasks; the dashed line denotes parity with autoregressive decoding.

draft path and the full R-recurrence path as verifier. The recurrent structure makes the reusable prefix explicit: we retain the state after recurrence r_d and continue verification through only the remaining $R - r_d$ applications of the shared core. No auxiliary draft model or additional parameters are introduced.

We use a margin-based stopping rule with threshold 0.1 and keep one fixed configuration per scale: $(r_d, R, K) = (3, 10, 8)$ for Small, $(1, 4, 4)$ for Medium, and $(2, 6, 8)$ for Large, where K is the maximum number of speculative tokens. The margin controls proposal length only; committed tokens are determined by the full-R verifier, so greedy decoding is lossless in exact arithmetic (Leviathan et al., 2023; Zhang et al., 2024). We generate 128 tokens from each task input and measure generation throughput. Across the nine downstream prompt distributions, Small spans $1.37\times$–$1.44\times$, Medium $1.18\times$–$1.39\times$, and Large $1.19\times$–$1.46\times$ speedup over full-depth autoregressive decoding (Figure 8); exact task-level throughput and acceptance are reported in Table 5.

C.2 Wall-Clock Latency and Decoding Memory

GRT executes the same number of block applications as the dense model it is *iso*FLOPs-matched to, distributed over fewer unique weight tensors. Table 6 reports measured generation latency and peak GPU memory on identical hardware (batch size 4, prompt length 1024, 128 generated tokens). Under `torch.compile`, GRT-Large costs $+10\%$ generation latency for 62% fewer parameters and 59% less peak memory; at medium scale the overhead is $+11\%$. In eager mode both overheads are $+23\%$, so roughly half the gap is kernel-launch overhead from the additional elementwise operations rather than arithmetic.

The $R\times$-expanded KV cache is a real cost, and whether it erodes the parameter saving depends on batch size. Table 7 reports end-to-end decoding memory (weights + KV, bf16, $T = 1024$) for the medium *iso*FLOPs checkpoint. At $B{=}1$ decoding is weight-dominated and the parameter reduction lands at $0.55\times$ dense memory even with the naive $R\times$ cache; at $B{=}32$ the cache dominates and the naive strategy recovers only $0.91\times$. Averaging K/V across recurrence steps brings this to $0.39\times$ while *improving* HellaSwag accuracy over the full cache (33.90 vs. 33.65), which suggests the averaging acts as a mild regulariser at this scale.

Table 5: **Task-level self-speculative decoding results.** Throughput is computed from the same aggregate wall-clock time per generated token used for the speedup, and is reported as autoregressive $\rightarrow$ self-speculative tokens/s. Acceptance is the fraction of draft tokens accepted by the full-R verifier. All runs use greedy decoding. Draft settings are fixed by scale: $(r_d, R, K) = (3, 10, 8)$ for Small, $(1, 4, 4)$ for Medium, and $(2, 6, 8)$ for Large.

	Small			Medium			Large		
Task	**AR→Spec.**	**Acc.**	**Speedup**	**AR→Spec.**	**Acc.**	**Speedup**	**AR→Spec.**	**Acc.**	**Speedup**
ARC-C	255.2→357.9	87.2%	1.40×	183.9→230.3	77.0%	1.25×	121.7→162.3	82.8%	1.33×
ARC-E	258.3→365.8	88.3%	1.42×	184.8→236.4	79.4%	1.28×	122.3→165.9	84.0%	1.36×
BoolQ	253.0→346.9	84.9%	1.37×	182.6→214.6	70.4%	1.18×	121.4→157.2	79.3%	1.30×
HellaSwag	254.7→357.3	87.5%	1.40×	184.8→217.9	72.4%	1.18×	121.4→144.8	72.9%	1.19×
LAMBADA-O	257.4→368.6	90.6%	1.43×	183.2→237.2	81.3%	1.29×	122.0→170.5	86.7%	1.40×
LAMBADA-S	254.6→366.5	91.6%	1.44×	184.5→227.6	77.6%	1.23×	121.5→162.7	83.6%	1.34×
OpenBookQA	254.6→349.5	87.8%	1.37×	183.8→232.2	79.2%	1.26×	122.3→164.8	84.8%	1.35×
PIQA	255.0→365.7	90.5%	1.43×	184.0→255.4	87.2%	1.39×	122.1→178.2	89.8%	1.46×
WinoGrande	255.3→349.4	88.2%	1.37×	184.2→224.3	75.8%	1.22×	121.5→161.6	84.0%	1.33×
HumanEval	247.6→300.5	79.3%	1.21×	170.4→234.0	84.6%	1.37×	105.4→132.7	81.1%	1.26×

Table 6: **Generation latency and peak memory (*iso*FLOPs configurations).** Batch size 4, prompt length 1024, 128 generated tokens, same hardware. Both arms are measured under `torch.compile` and in eager mode.

Model	**Config**	**Params**	**Eager (ms/tok)**	**Compiled (ms/tok)**	**Peak mem.**
GPT-2 Medium	24L	354M	2.95	2.54	747 MB
GRT Medium	`2+5×4+2`	127M	3.62	2.81	**402 MB**
GPT-2 Large	36L	774M	4.33	3.33	1570 MB
GRT Large	`1+5×6+5`	293M	5.33	3.67	**639 MB**

D Additional Ablations

D.1 Run-to-Run Variance Across Seeds

Table 8 reports three independent training runs (seeds 1337, 42, 123) of the small *iso*FLOPs configuration and of its dense counterpart, all at the full 20,000-step budget. GRT averages 3.145 ± 0.004 against 3.188 ± 0.056 for the dense baseline. GRT's worst seed (3.148) still falls below the dense baseline's best (3.154). Seed 1337 is the run reported throughout the main paper.

D.2 Component Ablation at Short Horizon

Table 9 reports the sequential ablation of Section 5 run on the *medium* configuration at 2,000 steps. At this horizon the ordering of the last two rows is inverted relative to the full 20,000-step run: prelude re-injection contributes -0.198 nats against the gate's -0.115. We report both because the comparison is itself informative — the structural components land early, whereas the gate is a learned mechanism whose contribution accrues over training.

D.3 Design Comparison of Recurrent-Depth Methods

Table 10 sets the small-scale results of Table 1 beside the design choices that produce them. MoR and Ouro gate the update, but Ouro supervises every iteration and MoR needs a router; heavy-tail Poisson shares strictly and updates unconditionally; RRT is input-independent and depth-fixed. GRT is the only column combining full sharing, an input-dependent update, and variable inference depth.

D.4 Gate Temperature Sensitivity

The gate temperature τ (Eq. 4) controls the sharpness of the sigmoid. We sweep $\tau \in \{0.5, 1.0, 2.0\}$ on the medium configuration at 5,000 steps. Validation loss is 3.62, 3.60, and 3.63 respectively, indicating that the optimal gate temperature is $\tau = 1.0$.

Table 7: **End-to-end decoding memory (medium *iso*FLOPs checkpoint, bf16, $T = 1024$).** Weights + KV cache. Parenthesised values are relative to GPT-2 Medium.

Configuration	HellaSwag	Mem. @ B=1	Mem. @ B=32
GPT-2 Medium (24L, 354M)	**34.45**	0.75 GiB (1.00×)	3.66 GiB (1.00×)
GRT *iso*FLOPs, full $R\times$ KV	33.65	0.41 GiB (0.55×)	3.32 GiB (0.91×)
GRT *iso*FLOPs, averaged KV	33.90	**0.35 GiB (0.47×)**	**1.44 GiB (0.39×)**

Table 8: **Three-seed validation loss on the small configuration** (20,000 steps). Both models use the *iso*FLOPs setting of Table 1. Lower is better.

Model	Seed 1337	Seed 42	Seed 123	Mean ± std
Dense baseline (12L)	3.154	3.156	3.253	3.188 ± 0.056
GRT (`1+1×10+1`)	**3.141**	**3.146**	**3.148**	$\mathbf{3.145 \pm 0.004}$

D.5 Noise Magnitude Sensitivity

We ablate state noise $\sigma_x \in \{0.0, 0.05, 0.1, 0.2\}$ on the medium configuration at 5,000 steps. Removing noise entirely ($\sigma_x = 0.0$) increases validation loss by 0.031 nats relative to $\sigma_x = 0.1$; larger noise ($\sigma_x = 0.2$) degrades by 0.018 nats. The optimal range is $\sigma_x \in [0.05, 0.1]$; we use 0.1 throughout.

D.6 Gate Bias and State Noise at Full Training Horizon

The sweeps above are at 5,000 steps. We repeated both at the full 20,000-step budget on the small configuration to check that the response surface does not sharpen at convergence. It does not. Moving the gate bias from the default $+4$ down to 0 costs 0.019 nats, and -2 recovers to 3.151; the surface is shallow and non-monotonic rather than peaked, and no setting produced a loss spike or divergence. State noise behaves the same way: removing it costs 0.018 nats and doubling it costs 0.019, a symmetric optimum rather than a value that has to be hit precisely. This is weaker evidence than a cross-architecture study, but it does suggest the mechanism does not depend on the exact bias value.

E Additional Qualitative Results

E.1 Per-Token Gate Activations Across Recurrence Steps

Figure 9 shows $\bar{g}_t^{(r)} = \mathrm{mean}_d(g_{t,:}^{(r)})$ — the elementwise gate averaged over the model dimension — for each token position t and recurrence step r, on four example prompts. Brighter (yellow-green) cells indicate write-heavy positions where the gate is open ($\bar{g} \approx 0.70$–0.82) and the shared block's proposal is substantially absorbed; darker (purple) cells indicate copy-heavy positions ($\bar{g} \approx 0.95$–1.00) where the hidden state passes through largely unchanged.

Three consistent observations emerge across all four prompts. First, **the per-step mean gate rises monotonically from step 2 onward**: means fall from ∼0.87–0.89 at step 1 to a minimum around step 2 (write-heavy phase), then recover steadily toward ∼0.88–0.91 at step 6 (copy-heavy phase). Second, **write-heavy positions are structurally consistent across steps**: the same columns that light up at step 1 tend to be the same at step 6, indicating that token identity — not recurrence depth — drives the gate primary signal, with the gate progressively dimming those writes over steps rather than shifting which tokens are written. Third, **content words and syntactically load-bearing tokens receive lower gate values than function words**: in "The old man sat by the", *man* and *sat* are visibly brighter than *the* and *by*; in the code prompt, the operator and identifier tokens show write-heavy behaviour while keywords like `def` and `for` are copy-heavy. This implicit routing — without any explicit per-token mechanism — arises purely from the gate conditioning on both the evolving hidden state $x^{(r)}$ and the fixed prelude anchor h.

F Broader Impact

Parameter-efficient language models have direct environmental and accessibility benefits: a GRT checkpoint at 37% of a baseline's parameter count requires proportionally less GPU memory at

Table 9: **Component ablation on GRT-medium at 2 000 steps.** Short-horizon counterpart to Figure 5. The dense baseline (row 1) is a standard 24-layer GPT trained identically.

	Configuration	Val. loss	Δ
1	Dense baseline (non-recurrent)	3.621	—
2	Recurrence	4.163	+0.542
3	+ Prelude / coda ($2+_{-}+2$)	4.118	−0.045
4	+ State noise (σ_x=0.1)	4.099	−0.019
5	+ Prelude re-injection	3.901	**−0.198**
6	+ Elementwise gate (full GRT)	**3.786**	−0.115

Table 10: **Design properties of recurrent-depth methods**, with small-scale *iso*FLOPS results for reference. Training times are wall-clock for the small configuration at 20,000 steps on identical hardware, and are *not* comparable against the dense column: the dense baseline was trained with `torch.compile` enabled, whereas every recurrent method including our own ran with compilation disabled, which did not work on our hardware. The dense column is therefore optimistic; the recurrent columns remain comparable to one another.

Property	Dense	MoR (Bae et al., 2025b)	Poisson (Geiping et al., 2025)	RRT (Bae et al., 2025a)	Ouro (Zhu et al., 2025a)	GRT (ours)
Full weight sharing	n/a	Yes	Yes	No (LoRA)	Yes	**Yes**
Variable inference depth	No	Yes	Yes	No	Yes	**Yes**
Input-dependent gating	No	Yes	No	No	Yes	**Yes**
Per-step noise injection	No	No	Init only	No	No	**Yes**
Per-iteration loss	No	No	No	No	Yes	No
Emergent early exit	No	Yes	Yes	No	Yes	**Yes**
Training time (small)	3h18m	12h19m	4h16m	6h42m	8h20m	6h18m
Small val. loss ↓	3.15	3.30	3.23	3.14	3.19	**3.14**

serving time, enabling deployment on hardware that cannot accommodate the full dense model and reducing energy consumption per inference step. Recurrent depth also provides a natural compute-quality tradeoff at inference time—the early-exit capability enabled by the depth-sampling procedure in Section 3 allows a single trained model to serve requests at multiple quality-FLOPs operating points without retraining.

The risks associated with GRT are not specific to its architecture: as a capable language model trained on web text, it inherits the standard dual-use risks of any such system, including the potential to generate misinformation or harmful content. These risks are addressed by standard deployment safeguards (content filtering, output monitoring, and responsible release practices) that apply equally to all current language models; the recurrent architecture does not introduce qualitatively new threat vectors.

G Recurrence Dynamics and KV Compression

Beyond aggregate validation loss, we probe *what* the shared block computes at each recurrence step $r \in \{1, \ldots, 6\}$ using the large $1{+}5{\times}6{+}5$ checkpoint. Diagnostics are run on a validation split with inference noise disabled. We report ten interlocking findings—grouped first into behavioural diagnostics (Section G.1–Section G.6) and then into mechanistic analyses of weights and attribution (Section G.7–Section G.10)—before drawing them together in Section G.12.

G.1 Rapid Convergence of Loss Across Recurrence Steps

Figure 10(a) traces the per-step validation loss as the shared block is applied iteratively. The prelude output alone yields a loss of 5.29; after a single application of the shared core the loss drops to 3.77—a reduction of 1.52 nats (natural-log-scale bits; 1 nat $= \log_2 e \approx 1.44$ bits) in one step. By step 2 the loss reaches 3.14, and the remaining four steps account for only an additional 0.46 nats, converging to the final value of 2.68. Stated differently, the first two recurrence steps account for roughly 77% of the total loss reduction from the prelude representation to the final output.

Table 11: Gate-bias sensitivity, small config at 20,000 steps.

Gate bias	Val. loss ↓
+4 (default)	**3.141**
+2	3.152
0	3.160
−2	3.151

Table 12: State-noise sensitivity, small config at 20,000 steps.

σ_x	Val. loss ↓	Δ
0	3.229	+0.018
0.1 (default)	**3.211**	—
0.2	3.230	+0.019

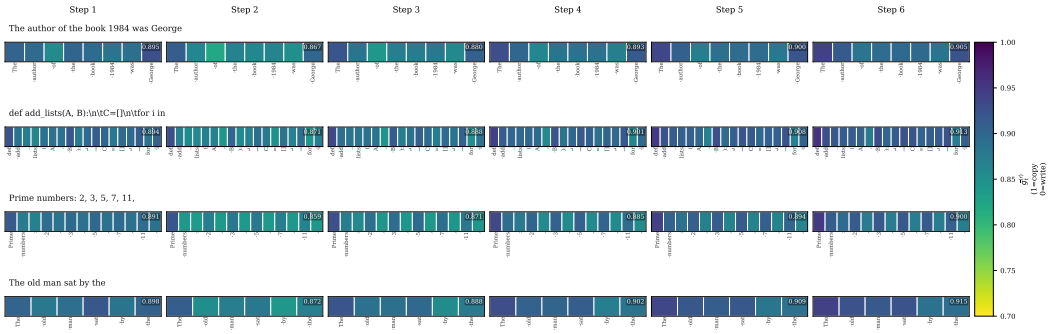


Figure 9: **Per-token gate activation $\bar{g}_t^{(r)}$ across recurrence steps.** Each row of cells corresponds to one recurrence step (columns = token positions). Colour encodes $\bar{g}_t^{(r)} \in [0.70, 1.00]$: bright yellow-green = write-heavy (gate open, block output substantially absorbed); dark purple = copy-heavy (hidden state passed through unchanged). Token labels are shown below each step column. The per-step mean (annotated top-right of each panel) rises from step 2 onward, reflecting the write-to-copy transition identified in Figure 11(b). Structurally informative tokens (content words, operators) consistently receive lower gate values than function words and punctuation across all steps.

This front-loaded profile is confirmed by the KL-divergence trajectory in Figure 10(b), which measures residual uncertainty relative to the final-step output distribution. The KL falls from 2.61 at the prelude to 0.46 by step 2 and reaches 0.014 at step 5, indicating that the output distribution is essentially committed after the fourth step. As a lower bound on the gate's utility, forcing the gate to zero throughout (the "no-recurrence" ablation) yields a catastrophic loss of 12.03, confirming that the iterative refinements are not redundant. The complementary ablation—forcing the gate to one throughout, so that every block proposal is discarded and the hidden state is never updated—yields a loss of 5.26, equivalent to the prelude-only output (5.26 at step $r = 0$). This confirms that recurrence with a fully open gate is vacuous: without selective writing, six recurrence steps reduce to a single prelude pass. Together, Gate=0 (12.03) and Gate=1 (5.26) bracket the trained model (2.68): the learned gate neither skips recurrence nor blindly overwrites state, but acquires a fine-grained elementwise balance between retention and update.

We also evaluate extending the recurrence depth at inference time beyond the $R = 6$ steps seen during training. Applying the trained shared block for $R \in \{8, 10, 12\}$ steps marginally *degrades* performance (2.69, 2.72, 2.74 respectively), suggesting the model has converged representationally by step 6 and that additional steps introduce noise rather than refinement.

Takeaway

77% of the loss reduction from prelude to final output occurs within the first two recurrence steps; the remaining four steps provide targeted corrections. The model's output distribution is essentially committed by step 4.

G.2 Gate Behaviour: Write-Heavy Early, Copy-Heavy Late

The elementwise gate $\mathbf{g}^{(r)} \in [0, 1]^{S \times d}$ controls how much of the shared block's proposal is written into the residual stream at each step. Figure 11(a) plots the per-step mean and standard deviation. The gate is most open—and most variable—in the middle steps (steps 2–3, mean ≈ 0.82, std ≈ 0.18),

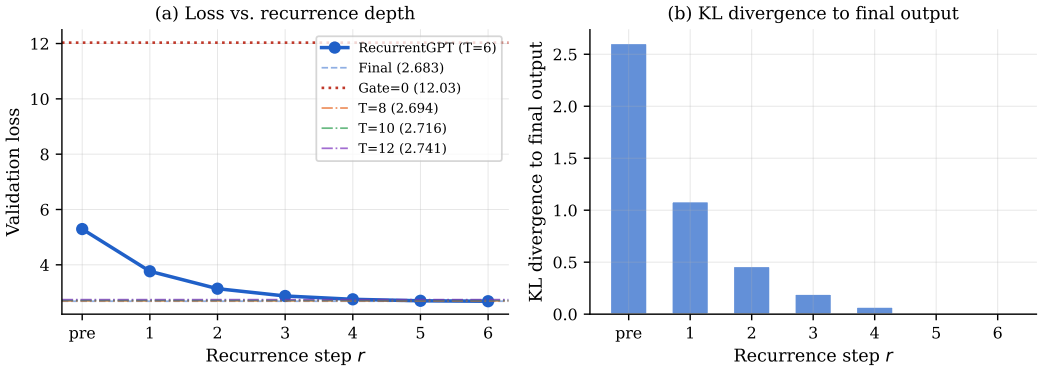


Figure 10: **Loss convergence across recurrence steps.** (a) Validation loss at each recurrence step r for the trained $1{+}5{\times}6{+}5$ model. The "Gate=1" baseline (loss 5.26, green dashed) equals the prelude-only output, confirming that selective writing is essential: a fully open gate discards every block proposal, leaving the hidden state unchanged across all six steps. The "Gate=0" baseline (loss 12.03) confirms that recurrence is essential, not a skip connection. Extended inference at $R \in \{8, 10, 12\}$ does not improve beyond $R = 6$ (2.69, 2.72, and 2.74 respectively), indicating representational convergence. (b) KL divergence of each step's output distribution to the final-step distribution; 95% of the residual uncertainty is resolved within four recurrence steps.

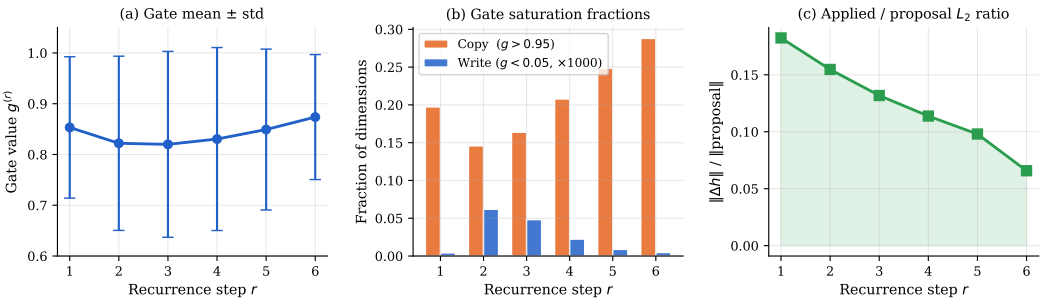


Figure 11: **Gate statistics across recurrence steps.** (a) Gate mean and standard deviation. (b) Fraction of copy-saturated ($g > 0.95$) and write-saturated ($g < 0.05$, scaled $\times 1000$) dimensions per step. Copy-saturation grows steadily while write-saturation stays negligible. (c) Effective gate openness (applied/proposal L_2 ratio), declining monotonically—early steps write substantially, late steps refine conservatively.

corresponding to the highest information-writing regime, and tightens progressively toward step 6 (mean 0.87, std 0.12).

Figure 11(b) quantifies this transition via the fraction of dimensions where the gate is near-saturated. Copy-saturated dimensions ($g > 0.95$) grow monotonically from 19.7% at step 1 to 28.8% at step 6, while write-saturated dimensions ($g < 0.05$) remain negligibly rare ($< 10^{-4}$) throughout. This asymmetry—abundant copying, scarce full-overwriting—indicates the model prefers selective blending over clean state replacement.

The effective gate openness (Figure 11(c)), measured as the ratio

$$\rho^{(r)} = \frac{\|\mathbf{h}^{(r)} - \mathbf{h}^{(r-1)}\|_2}{\|\mathbf{o}^{(r)} - \mathbf{h}^{(r-1)}\|_2} = \frac{\|(1 - \mathbf{g}^{(r)}) \odot (\mathbf{o}^{(r)} - \mathbf{h}^{(r-1)})\|_2}{\|\mathbf{o}^{(r)} - \mathbf{h}^{(r-1)}\|_2}, \tag{10}$$

where the numerator is the norm of the *applied update* (actual change to the hidden state) and the denominator is the norm of the *proposal* (the full update the block would write if the gate were fully open), decreases monotonically from 0.182 at step 1 to 0.066 at step 6. Taken together, the gate transitions the model from a *write-heavy* regime in early steps—where large fractions of the proposal are incorporated to rapidly restructure the representation—to a *copy-heavy* regime in later steps, where the representation is largely preserved and only small targeted corrections are applied.

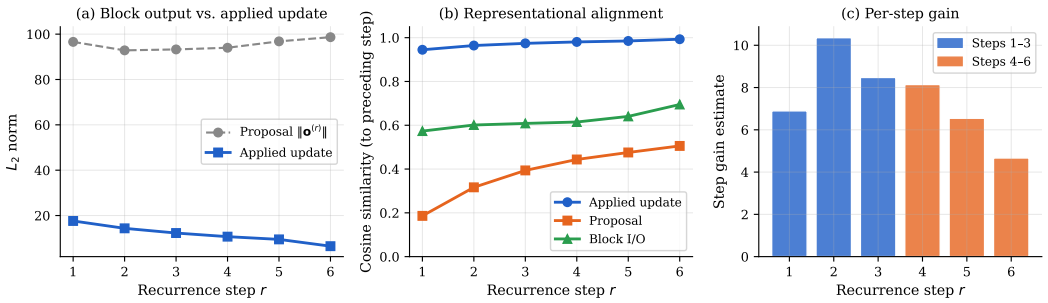


Figure 12: **Update dynamics across recurrence steps.** (a) L_2 norms of the block's raw proposal (near-constant, ≈ 94) and the gate-applied update (declining $17.6 \rightarrow 6.5$); the divergence is caused entirely by gate closure. (b) Cosine similarity between successive representations. (c) Local step gain estimate, peaking at step 2 and declining thereafter.

Takeaway

The gate transitions from a write-heavy regime (steps 1–2, mean ≈ 0.82) to a copy-heavy regime (steps 3–6, mean ≈ 0.87), with effective gate openness declining monotonically from 0.18 to 0.07.

G.3 Update Dynamics: Diminishing Magnitude, Increasing Alignment

Figure 12(a) separates two quantities at each step: the L_2 norm of the shared block's raw proposal $\mathbf{o}^{(r)}$, and the L_2 norm of the gate-applied update that actually enters the residual stream. The proposal norms are nearly constant across all six steps (93–99), possibly indicating that the shared block does not itself "know" it is being applied repeatedly—its output magnitude is stationary. The applied update, by contrast, declines from 17.6 at step 1 to 6.5 at step 6, driven entirely by the gate becoming more closed.

Figure 12(b) shows cosine similarity between successive representations, $\cos_\text{sim}(\mathbf{h}^r, \mathbf{h}^{r-1}) = \frac{\mathbf{h}^r \cdot \mathbf{h}^{r-1}}{\|\mathbf{h}^r\| \|\mathbf{h}^{r-1}\|}$. The proposal cosine similarity grows from 0.19 at step 1 to 0.51 at step 6, indicating that the block and the hidden state increasingly agree on direction as the representation stabilises. The applied update cosine similarity is already high at step 1 (0.94) and approaches 0.99 by step 6, confirming that late-step corrections are directionally consistent refinements.

The per-step gain estimate (Figure 12(c)) peaks at step 2 (10.4), declining to 4.7 at step 6. This concave gain profile, in which the most productive computation occurs before representational alignment is achieved, is consistent with the loss convergence plots in Section G.1.

G.4 Representational Geometry: Two-Phase Structure

We use Centered Kernel Alignment (CKA; Kornblith et al., 2019) to compare hidden-state representations across recurrence steps and against the corresponding layer representations of the GPT-2 Large dense baseline.

Within-model CKA (Figure 13(a)). The 7×7 CKA matrix reveals a two-phase structure. Representations at step 0 (prelude) and step 1 are nearly identical (CKA $= 0.996$), while both are sharply dissimilar from all subsequent steps (CKA ≈ 0.33–0.65 with steps ≥ 2). Steps 2 through 6 form a tightly cohesive cluster (pairwise CKA ≥ 0.907), within which similarity decreases only gradually with step distance. This suggests the model undergoes a *representational phase transition* between steps 1 and 2.

Cross-model CKA (Figure 13(b)). Prelude and step 1 representations align maximally with the shallow layers of the baseline (peak at layer 2, CKA $= 0.993$), while steps 2 through 6 all peak at baseline layer 11 (CKA $= 0.97$–0.90 declining). No recurrent step aligns strongly with the deep layers of the baseline (layers > 20, CKA < 0.50), confirming that recurrent depth compresses the computational path but does not replicate the later representational hierarchy of the dense model.

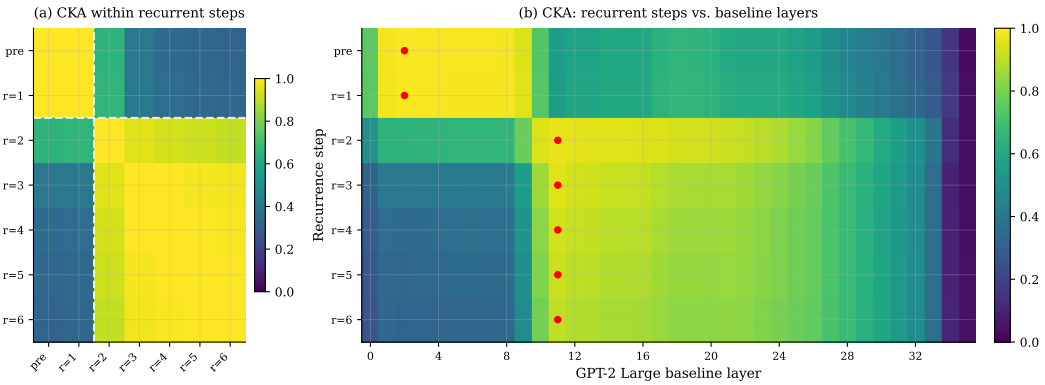


Figure 13: **CKA representational geometry.** (a) Within-model CKA; the sharp discontinuity between steps 1–2 and steps 3–6 (dashed lines) reveals a two-phase structure. (b) Cross-model CKA against GPT-2 Large; early steps map to shallow baseline layers, all later steps peak at layer ≈ 11.

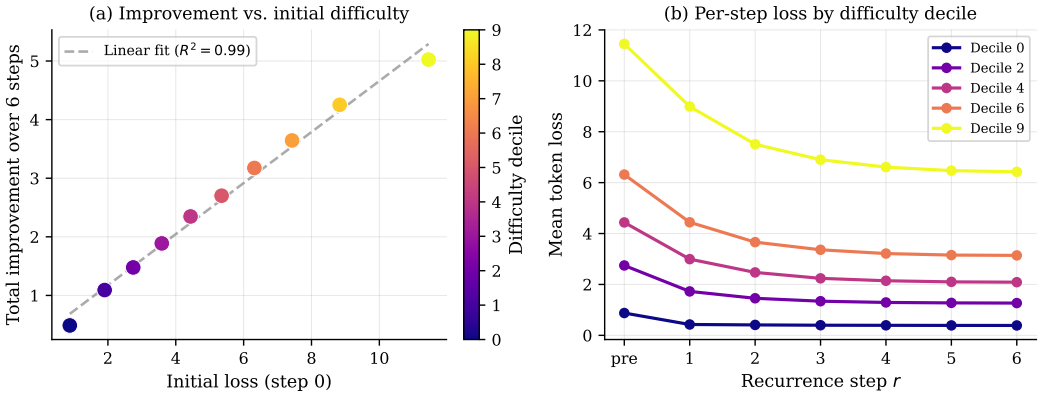


Figure 14: **Token difficulty routing.** (a) Total improvement over six steps vs. initial difficulty decile ($R^2 = 0.998$); recurrence implicitly allocates more work to uncertain tokens. (b) Per-step loss curves for selected deciles.

Takeaway

A representational phase transition occurs between steps 1 and 2: early steps resemble shallow baseline layers, while steps 2–6 all align with mid-depth baseline layer 11, compressing the computational path without replicating the deep-layer hierarchy.

G.5 Differential Routing by Token Difficulty

To test whether the gate allocates computation to where it is most needed, we sort tokens by their loss after the prelude (step 0) into ten deciles and track the total loss improvement accumulated over six recurrence steps. Figure 14(a) shows a near-linear relationship between initial token difficulty and total improvement ($R^2 = 0.998$): the easiest decile gains only 0.49 nats, while the hardest decile gains 5.02 nats—a ten-fold difference. The gate difficulty correlation—whether tokens with higher loss also receive lower (more open) gate values—is mildly negative across all steps (-0.07 to -0.04), consistent with the gate allowing slightly more writing for uncertain tokens.

G.6 Fixed-Point Convergence and Effective Rank

Figure 15(a) tracks the effective rank of the hidden-state tensor at each step. The rank increases from 0.546 at the prelude to 0.635 by step 3, after which it plateaus, indicating that iterative updates progressively spread information across more hidden dimensions. The effective rank of the *delta*

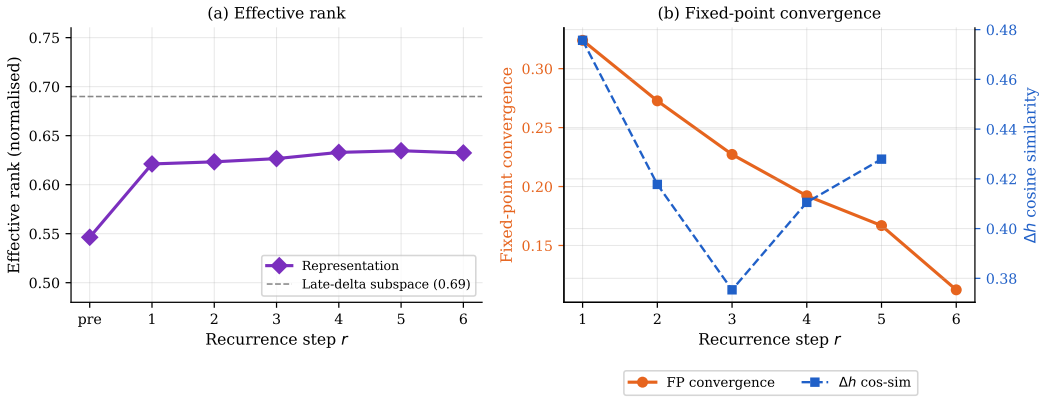


Figure 15: **Effective rank and fixed-point convergence.** (a) Effective rank grows with depth; the late-delta subspace rank (dashed) is higher than the representation itself. (b) Fixed-point convergence (orange) declines monotonically; Δh cosine similarity (blue) increases—both consistent with convergent refinement.

subspace—the directions along which the hidden state changes in later steps—is 0.69, higher than the rank of the representation itself, suggesting that late-step updates refine a diverse set of features.

Figure 15(b) plots the normalised fixed-point convergence metric, defined as

$$\delta_{\text{fp}}^{(r)} = \frac{\|\mathbf{h}^{(r)} - \mathbf{h}^{(r-1)}\|_F}{\|\mathbf{h}^{(r-1)}\|_F}, \tag{11}$$

the Frobenius-norm magnitude of the update relative to the current representation, which declines monotonically from 0.324 at step 1 to 0.112 at step 6, confirming that the recurrent dynamics are contractive.

Table 13 reports a sanity check via anchor ablations—swapping the fixed prelude output $\mathbf{h}^{(\text{pre})}$ fed at each recurrence step for alternative anchors while keeping the gate and blocks identical. Crucially, the "previous hidden state" row uses $\mathbf{h}^{(r-1)}$ as anchor (i.e. the anchor drifts with the recurrence rather than being held fixed), while the learned gate remains fully active and adaptive. The 0.70-nat gap between this condition (3.38) and the trained model (2.68) therefore quantifies the value of providing a *stable, fixed reference point* at every step, not merely the value of the gate.

Table 13: Anchor ablation: effect of replacing $\mathbf{h}^{(\text{pre})}$ with alternative anchors at each recurrence step. The "prev. hidden state" anchor drifts at each step ($\mathbf{h}^{(r-1)}$) while the learned adaptive gate remains active throughout. The 0.70-nat gap between that condition and the trained model isolates the value of holding the prelude output *fixed* as a stable context reference.

Anchor type	**Val. loss (nats)**	Δ vs. trained
Zeros	8.08	+5.40
Input embedding	3.73	+1.05
Prev. hidden state $\mathbf{h}^{(r-1)}$	3.38	+0.70
Trained model (fixed $\mathbf{h}^{(\text{pre})}$)	**2.68**	—

G.7 Gate Attribution: Current State Gradually Supersedes the Anchor

The elementwise gate (Eq. 4) reads from two sources simultaneously: $\hat{\mathbf{x}}^{(r)} \triangleq \text{LN}(\mathbf{h}^{(r-1)})$, the LayerNorm-normalised current hidden state,[1] and $\hat{\mathbf{h}} \triangleq \text{LN}(\mathbf{h}^{(\text{pre})})$, the normalised prelude anchor. The gate MLP $f_{\mathbf{g}}$ takes their concatenation as input; we write its first linear layer weight as $W_{\mathbf{g}} = [W_x \mid W_h] \in \mathbb{R}^{d_{\text{gate}} \times 2d}$, where $W_x \in \mathbb{R}^{d_{\text{gate}} \times d}$ acts on $\hat{\mathbf{x}}^{(r)}$ and $W_h \in \mathbb{R}^{d_{\text{gate}} \times d}$ acts on $\hat{\mathbf{h}}$. To quantify which source drives the gate at each step, we measure how much variance in the actual per-token gate

[1]We write $\hat{\mathbf{x}}^{(r)}$ rather than x to avoid confusion with token IDs $(x_1, \ldots, x_S)$ introduced in the Preliminaries.

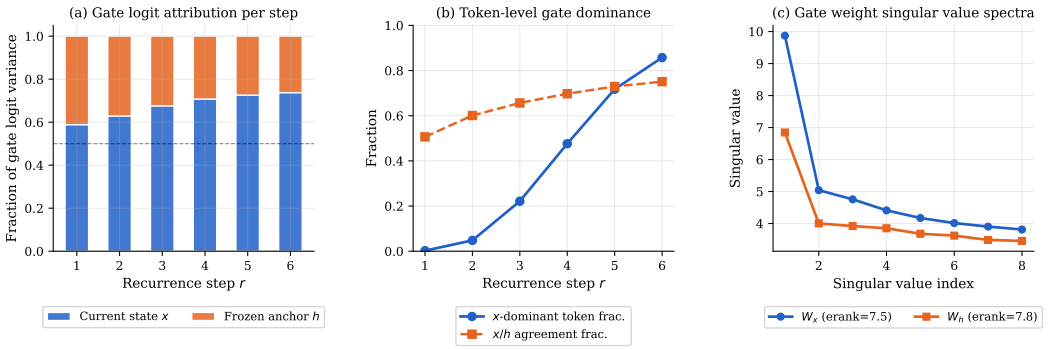


Figure 16: **Gate attribution across recurrence steps.** (a) Fraction of gate logit variance from current state x (blue) vs. frozen anchor h (orange); the crossover from rough parity at step 1 to 74%/26% at step 6 is explained by growing divergence between $x^{(r)}$ and h. (b) Token-level x-dominance and x/h sign-agreement fractions. (c) Gate weight singular value spectra; the 44% gap in leading singular values (9.87 vs 6.85) is the structural origin of the state's eventual dominance.

logit each half explains. As seen in Figure 16(a), at step 1 the split is nearly even: 59% current state, 41% anchor. By step 6 this shifts substantially—74% current state, 26% anchor.

Figure 16(b) shows the same trend at the token level: the fraction of positions where the state half produces a larger absolute gate logit than the anchor grows from a near-zero 0.2% at step 1 to 86% at step 6. The answer to *why* is present in the weights before any forward pass: we find in Figure 16(c) that the leading singular value of W_x is 9.87 versus 6.85 for W_h, giving the state half structurally greater sensitivity. At step 1 the hidden state is close to the prelude output, so the gap is small; as $x^{(r)}$ diverges from h across steps, this structural asymmetry is amplified into the monotone attribution shift in Figure 16(a).

Takeaway

By step 6, the gate reads 74% of its signal from the current hidden state and only 26% from the fixed prelude anchor. This shift is structurally predetermined: W_x has a 44% larger leading singular value than W_h, priming the gate to amplify state-driven signals as $\mathbf{h}^{(r)}$ departs from $\mathbf{h}^{(\text{pre})}$.

G.8 Projection Weight Analysis: State and Anchor as Complements

Before the shared blocks run, the recurrent projection, W_{proj}, maps the concatenated $[x^{(r)}, h]$ into the d-dimensional input the transformer sees. A natural question is whether this projection simply averages the two sources or treats them as carrying distinct information. The answer is unambiguous: the mean row-wise cosine similarity between the W_x and W_h halves is $-\mathbf{0.189}$, with only 0.16% of the 1024 rows aligned above a cosine of 0.5.

Both halves are nearly full-rank (effective ranks 1002.9 and 1004.3, Frobenius norms 32.7 and 23.6), with flat, slowly-decaying singular value spectra (Figure 17(a)–(b)). The negative mean row cosine means the projection reads the *difference* between current state and anchor rather than their average; we call this the *contrastive projection* property. Directions large in both $x^{(r)}$ and h—stable features unchanged since the prelude—partially cancel; directions large only in $x^{(r)}$ are amplified. This provides a direct mechanistic explanation for why the fixed prelude is helpful: it supplies the reference needed to compute what has changed.

G.9 Attention Entropy: Stable Head Specialisations, Gradual Sharpening

Because identical weights govern every recurrence step, the shared block cannot self-modulate its attention patterns—any change must arise purely from changes in $\mathbf{h}^{(r)}$. In Figure 18(a), we capture the full $[B, n_{\text{head}}, S, S]$ attention tensors at each step (where B is batch size and n_{head} the number

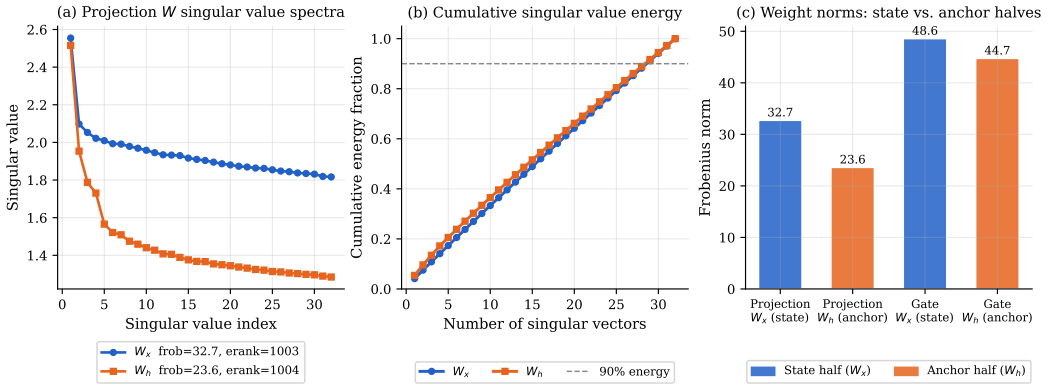


Figure 17: **Recurrent projection weight analysis.** (a) Singular value spectra of W_x and W_h; both are nearly full-rank. (b) Cumulative energy; both halves reach 90% at similar dimensionalities. (c) Frobenius norms of all four weight halves; the state half (W_x) consistently outweighs the anchor half (W_h) in both modules.

of attention heads) across all five shared blocks (100 heads total) and measure per-head Shannon entropy and diagonal mass.

Aggregate trend. Mean attention entropy declines from 3.00 nats at step 1 to 2.77 at step 4, then *rebounds* to 2.82 at steps 5–6 (Figure 18(b)). This non-monotone profile mirrors the two-phase CKA structure from Section G.4: steps 1–4 are the refinement phase (lower entropy, more peaked patterns), while the rebound at steps 5–6 aligns with the final convergence plateau where small corrective updates scan broadly. Mean diagonal mass peaks at step 4 (0.108) then drops to 0.092 at step 6.

Head role stability. The full 6×100 entropy heatmap (Figure 18(a)) makes the dominant pattern immediately visible: individual heads retain their character across all six steps. Of 100 heads, 2 are local/sharp (entropy < 1.0), 3 are previous-token (prev-mass > 0.35), 2 are self-attention (diagonal mass > 0.35), 7 are broad broadcast (entropy > 4.0), and 86 are general. Here, *prev-mass* denotes the fraction of total attention weight placed on the immediately preceding token position (i.e. diagonal offset -1 of the attention matrix), averaged over the sequence. A head with prev-mass > 0.35 predominantly attends to the token immediately before each query position. The clearest specialist—Block 0, Head 14—has entropy 0.298 and prev-mass 0.878 at step $r = 1$,

Takeaway

Head specialisations are locked in by weight and do not adapt to recurrence depth. What changes across steps is only the magnitude of each head's contribution, modulated by the gated update to the hidden state.

G.10 Recurrence Preferentially Serves Two Token Regimes

We split tokens into four BPE frequency bands: byte/punctuation (IDs 0–255, $n = 5{,}763$), common (IDs 256–2,000, $n = 15{,}789$), medium (IDs 2,001–10,000, $n = 6{,}778$), and rare (IDs $> 10{,}000$, $n = 4{,}438$), where n refers to the total occurrences of corresponding token id ranges in the validation set. Figure 19(a) shows that all four follow the familiar front-loaded loss-reduction profile, but at very different absolute scales: byte/punctuation tokens improve by just 1.92 nats while rare tokens improve by 3.39 nats.

Figure 19(b) plots the per-step relative improvement $\Delta_r^{\text{rel}} = (L^{(r-1)} - L^{(r)})/L^{(r-1)}$, i.e. the fraction of the *preceding step's* loss resolved at each step. At step $r = 1$, byte/punctuation tokens already resolve 49% of their previous loss, compared to 23–28% for the other bands—recurrence is disproportionately productive for easy tokens first. Gains diminish uniformly beyond step 3 ($< 4\%$ for all bands), consistent with the fixed-point convergence of Section G.6. Cumulatively over all six steps, byte/punctuation achieves 67% recovery of its initial loss versus 42% for rare tokens. We call these two roles *completion* (sharpening already-likely token predictions) and *lexical narrowing*

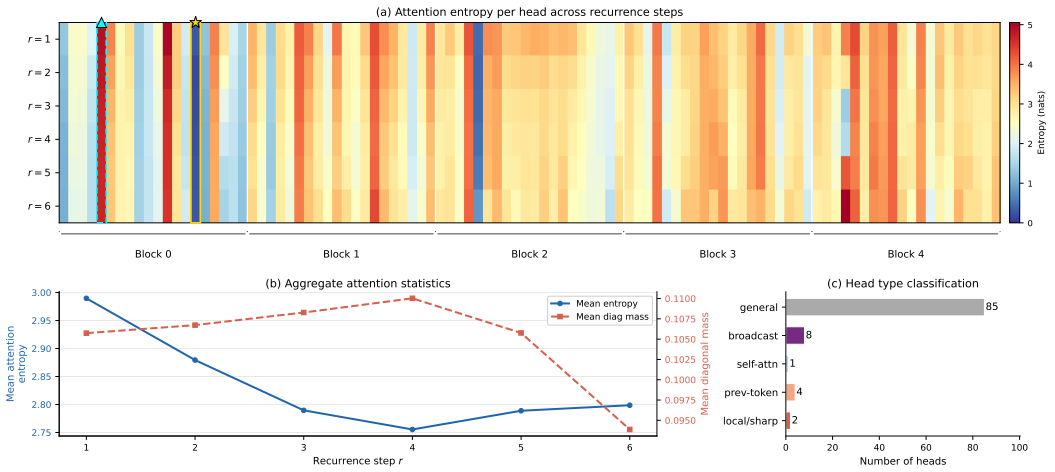


Figure 18: **Attention entropy across recurrence steps.** (a) Full 6×100 entropy heatmap; white lines separate blocks. $\star$: Block 0 Head 14 (previous-token specialist, entropy 0.30). $\triangle$: Block 0 Head 4 (broadcast head, entropy 4.87). Head roles are highly stable. (b) Mean entropy (blue) and diagonal mass (orange) per step; non-monotone entropy profile mirrors the two-phase CKA structure. (c) Head type classification across all 100 heads.

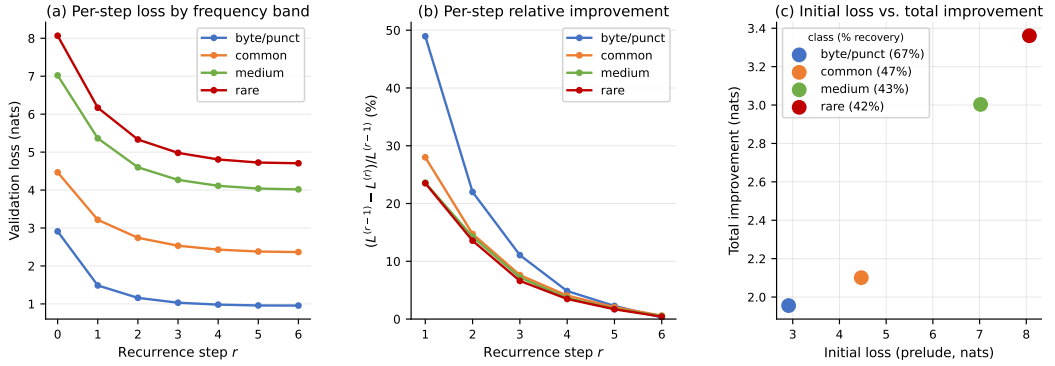


Figure 19: **Token frequency class breakdown.** (a) Per-step loss by BPE frequency band; rare tokens maintain persistently higher loss throughout. (b) Per-step relative improvement $(L^{(r-1)} - L^{(r)})/L^{(r-1)}$ per frequency class; the first recurrence step alone resolves 47% of byte/punctuation loss vs. 23% for rare tokens—easy tokens converge fastest, hard tokens accumulate improvements gradually. (c) Initial loss vs. total improvement; the two axes are near-orthogonal, consistent with dual completion/narrowing roles.

(progressively concentrating probability mass for hard tokens). Both operate simultaneously in the same forward pass, possible because the elementwise gate allocates computation per-position rather than globally—no explicit routing mechanism is required.

G.11 Recurrent KV Cache Dynamics

The write-to-copy transition in Figure 11 is visible in the attention cache as well. We analyse the small `1+1×10+1` checkpoint because its ten recurrent steps provide the finest temporal resolution.

Keys remain stable while values evolve. Let $K^{(r)}$ and $V^{(r)}$ denote the recurrent key and value states at step r. Figure 20(a) measures their relative displacement from recurrence 1. By recurrence 10, the relative K displacement is only 0.28, compared with 0.83 for V. The same separation appears in cosine similarity (Figure 20(b)): K stays above 0.98 across the trajectory, while V falls to 0.64. Most cache evolution therefore occurs in V, while K preserves a comparatively stable addressing geometry.

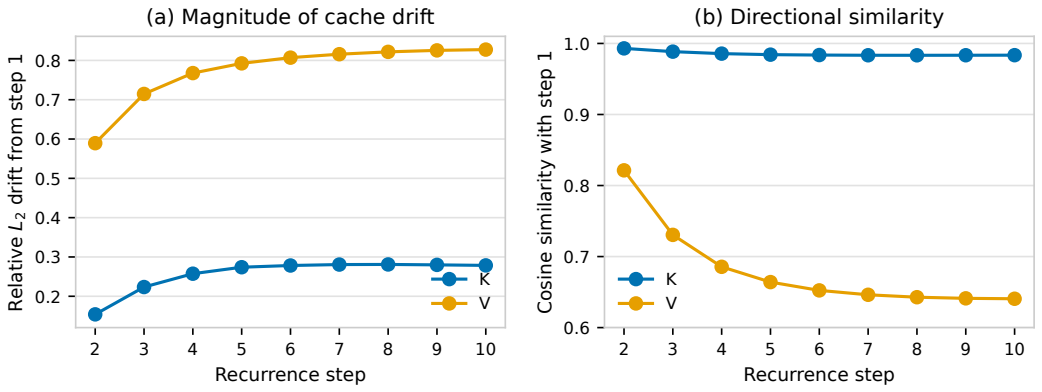


Figure 20: **K/V evolution across recurrence.** Relative to recurrence 1, K remains numerically and directionally stable, whereas V changes rapidly during the early recurrent steps before saturating.

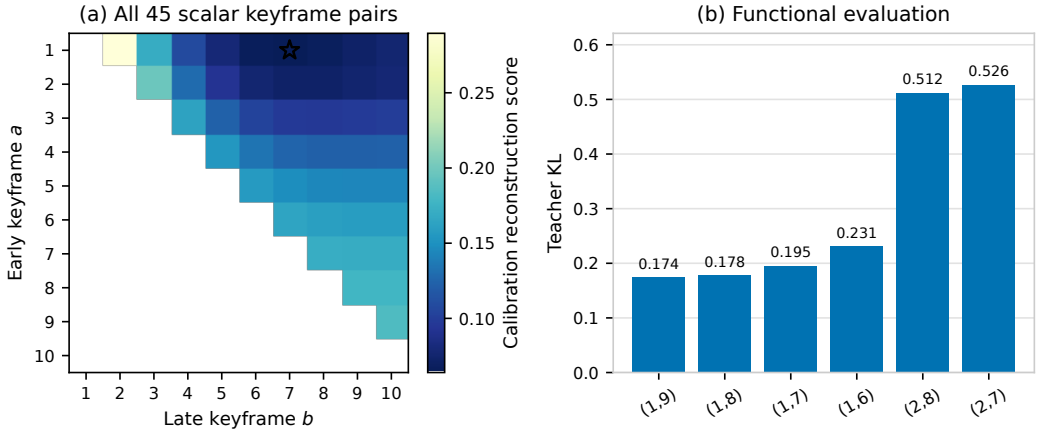


Figure 21: **Pairwise recurrence interpolation.** (a) Reconstruction score for all 45 pairs (a, b), where a and b identify two recurrence steps whose cache states are retained for the interpolation. The magenta star marks the lowest reconstruction error. (b) Teacher KL for the strongest scalar-interpolation candidates; functional fidelity favours a later second state than reconstruction alone.

Temporal complementarity. To quantify which recurrence positions contain complementary cache information, for every pair (a, b) with $1 \le a < b \le R$ we retain the two states $C^{(a)} = (K^{(a)}, V^{(a)})$ and $C^{(b)} = (K^{(b)}, V^{(b)})$. For each target recurrence r, scalar coefficients are fitted on calibration tokens:

$$\widehat{K}^{(r)} = \lambda_r^K K^{(a)} + (1 - \lambda_r^K) K^{(b)}, \tag{12}$$

$$\widehat{V}^{(r)} = \lambda_r^V V^{(a)} + (1 - \lambda_r^V) V^{(b)}. \tag{13}$$

The coefficients $\lambda_r^K, \lambda_r^V \in [0, 1]$ are shared across tokens and heads but fitted independently for K and V. We score each pair by the recurrence-averaged relative MSE, $0.25\,\mathrm{RelMSE}(K) + \mathrm{RelMSE}(V)$.

With $R = 10$, this gives all $\binom{10}{2} = 45$ recurrence pairs in Figure 21(a). Low reconstruction error concentrates along pairs that combine recurrence 1 with a substantially later state; nearby recurrences are less complementary. The best reconstruction pair is $(1, 7)$. We then evaluate the strongest candidates by replacing the recurrent history with their interpolated caches and measuring teacher KL. The functional criterion shifts the late state to recurrence 9: $(1, 9)$ reaches KL 0.174, followed by $(1, 8)$ at 0.178 and $(1, 7)$ at 0.195 (Figure 21(b)). The information needed for the model's output therefore continues to change after the raw cache trajectory is already well approximated.

For the functionally preferred $(1, 9)$ pair, the fitted interpolation follows a smooth early-to-late schedule. The weight on recurrence 1 drops from approximately one at $r = 1$ to $0.47/0.58$ for K/V at $r = 2$, $0.22/0.34$ at $r = 3$, and is effectively zero by $r = 8$–9 (Figure 22). This closely matches

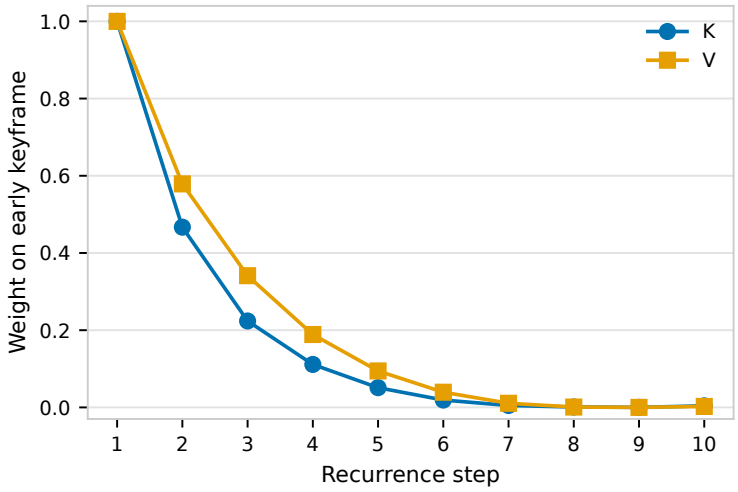


Figure 22: **Early-to-late cache schedule.** Recurrence-dependent interpolation weights on the recurrence-1 state for the functionally preferred $(1, 9)$ pair.

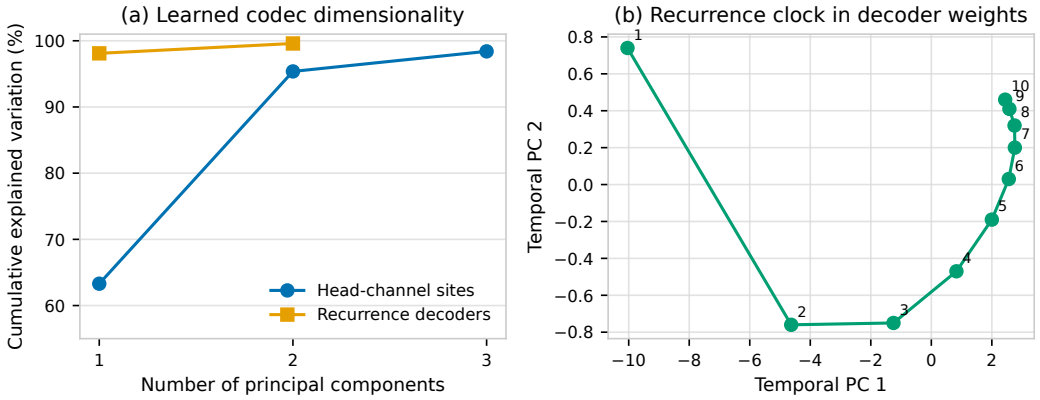


Figure 23: **Structure of the learned Channel-wise KV Codec.** (a) Cumulative explained variation across head-channel sites and recurrence-specific decoders. (b) The first two temporal principal coordinates of the decoder weights; labels denote recurrence steps.

the gate dynamics: large updates occur early, followed by progressively smaller changes toward a late cache state.

Low-dimensional channel structure. The learned parameters of the Channel-wise KV Codec (Appendix B) are themselves highly structured. On the small model, flattening the encoder and decoder weights for each of the $12\times64 = 768$ head-channel sites shows that two principal components explain 95.37% of variation across sites and three explain 98.4%. Recurrence dependence is even lower-dimensional: one temporal component explains 98.10% of decoder variation and two explain 99.59%. The corresponding two-dimensional temporal trajectory (Figure 23(b)) moves most strongly over the first few recurrences and then contracts. Thus both the raw cache trajectory and the learned compressor expose the same rapidly saturating recurrence clock.

Takeaway

Recurrent cache evolution is strongly asymmetric and temporally structured: K remains close to its initial recurrent state, V carries most of the change, and the trajectory moves rapidly from an early state toward a stable late state. The same low-dimensional recurrence clock appears in the learned codec.

G.12 Unified Summary

The analyses above converge on a coherent computational and mechanistic account of how GRT's shared block achieves its parameter efficiency.

Behaviourally, the model operates in two regimes separated by a representational phase transition between steps 1 and 2. In the *early regime* (steps 1–2), the gate is relatively open (≈ 0.82 mean, large update norms, high proposal cosine entropy), the hidden state undergoes a rapid restructuring that resolves $\sim$77% of predictive uncertainty, and CKA confirms a sharp departure from the prelude representation. In the *late regime* (steps 3–6), the gate progressively closes, updates shrink and align, the representation approaches a fixed point, and the attention patterns partially rebroadcast—consistent with targeted correction rather than coarse restructuring.

Mechanistically, four findings explain *why* this behaviour emerges from the learned weights. The *gate weight asymmetry* (W_x leading singular values 44% larger than W_h) primes the gate to respond more strongly to changes in the hidden state than to the anchor, an advantage that is dynamically amplified as $x^{(r)}$ departs from h across steps. The *contrastive projection* (mean row cosine -0.19) ensures the shared block receives an input that emphasises what has changed since the prelude, giving each step access to a difference signal rather than a raw state. The *head specialisations* are fixed by weight—the same 14 previous-token heads, 8 broadcast heads, and 11 self-attention heads appear at every step—so the block applies the same functional program each time; the gate, not the heads, decides how much of each step's output to incorporate. The *token frequency breakdown* reveals that a single forward pass simultaneously performs completion for common tokens and lexical narrowing for rare ones, made possible because the elementwise gate implicitly routes computation per-position without any explicit routing mechanism. The recurrent KV cache shows the same temporal organisation: K remains comparatively stable, while V carries a larger early-to-late transition that is well described by a low-dimensional recurrence clock.