Memory-Efficient Looped Transformer: Decoupling Compute from Memory in Looped Language Models

Anonymous Authors[1]

Abstract

Recurrent LLM architectures have emerged as a promising approach for improving reasoning, as they enable multi-step computation in the embedding space without generating intermediate tokens. Models such as Ouro (Zhu et al., 2025) perform reasoning by iteratively updating internal representations while retaining a standard Key-Value (KV) cache across iterations, causing memory consumption to grow linearly with reasoning depth. Consequently, increasing the number of reasoning iterations can lead to prohibitive memory usage, limiting the practical scalability of such architectures. In this work, we propose **Memory-Efficient Looped Transformer (MELT)**, a novel architecture that decouples reasoning depth from memory consumption. Instead of using a standard KV cache per layer and loop, MELT maintains a single KV cache per layer that is shared across reasoning loops. This cache is updated over time via a learnable gating mechanism. To enable stable and efficient training under this architecture, we propose to train MELT using *chunk-wise training* in a two phase procedure: *interpolated transition*, followed by *attention-aligned distillation*, both from the LoopLM starting model to MELT. Empirically, we show that MELT models fine-tuned from pretrained Ouro parameters **outperform standard LLMs of comparable size**, while maintaining a memory footprint comparable to those models and dramatically smaller than Ouro's. Overall, MELT achieves constant-memory iterative reasoning without sacrificing LoopLM performance, using only a lightweight post-training procedure.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

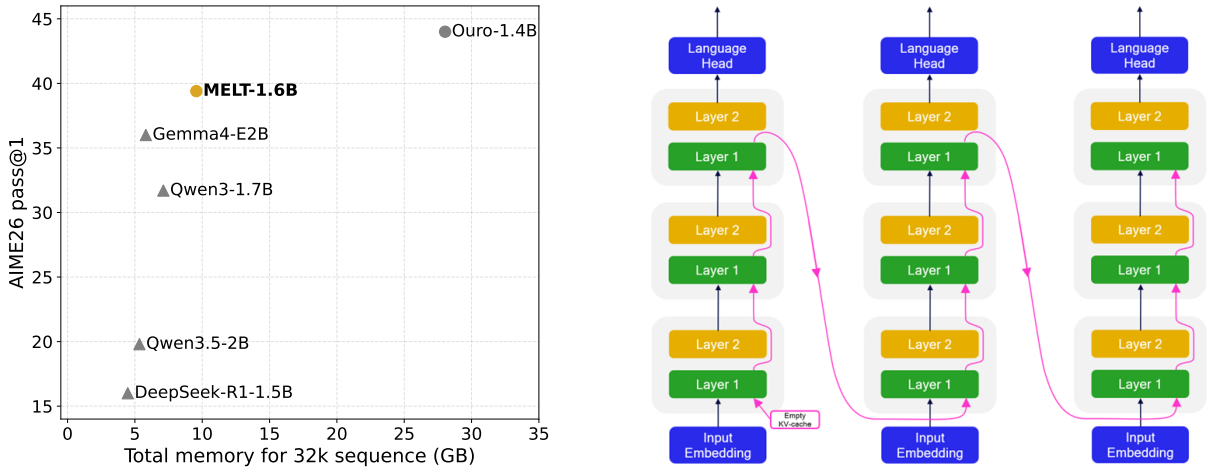
1. Introduction

Large Language Models (LLMs) increasingly rely on inference-time compute to improve reasoning, shifting away from purely scaling training-time compute. A dominant approach is Chain-of-Thought (CoT) prompting, where models generate intermediate "thinking" tokens before producing a final answer. While effective, this couples reasoning depth to output length, increasing latency and memory usage. An alternative is latent reasoning, where models perform additional internal computation without generating extra tokens.

A prominent instantiation of latent reasoning is looped transformers, which perform recurrence at the architecture level by repeatedly passing hidden states through the same transformer stack. This approach was first explored in Universal Transformers (Dehghani et al., 2019) and has recently shown impressive gains with LoopLM (Zhu et al., 2025), demonstrating that looped models can match or surpass transformers nearly twice their size. However, these approaches suffer from a key limitation: memory grows linearly with the number of loops due to Key-Value (KV) states. To address this, we propose **Memory-Efficient Looped Transformer (MELT)**, which decouples reasoning depth from memory consumption by maintaining a single KV entry per token and layer, updated across loops via a learnable gating mechanism. This design preserves full attention while keeping memory usage fixed as iterative depth increases.

We demonstrate this approach by training a MELT model initialized from pretrained Ouro (Zhu et al., 2025) weights. Empirically, we show that MELT outperforms similarly sized standard transformers on reasoning benchmarks while preserving the performance of the originating LoopLM, but with dramatically lower memory than looped baselines that retain per-loop KV growth. The main contributions of this paper are:

- We introduce **MELT**, a memory-efficient looped transformer architecture that decouples reasoning depth from memory consumption by sharing a single KV-cache per layer across reasoning loops and updating it with a learnable gating mechanism.
- We propose a data-efficient procedure for adapting pre-



a) MATH-500 accuracy versus memory usage

b) High-level illustration of MELT

Figure 1. **(a)** MELT achieves superior performance compared to similarly sized non-looped models, while maintaining an equivalent memory footprint, only slightly higher due to the absence of MQA. **(b)** As in looped transformers, layers are reused across iterations, but the KV cache is updated rather than expanded across loops.

trained LoopLMs to MELT through **chunk-wise training** and a two phase procedure: (i) **interpolated transition** from LoopLM to MELT and (ii) **attention-aligned distillation** using the frozen LoopLM as a layer-wise teacher to consolidate the learned representations.

- We empirically show that a MELT model initialized from pretrained Ouro parameters **outperforms standard LLMs** of comparable size, while matching their memory footprint and using substantially less memory than Ouro.

All the code to replicate our experiments and the model itself will be released soon.

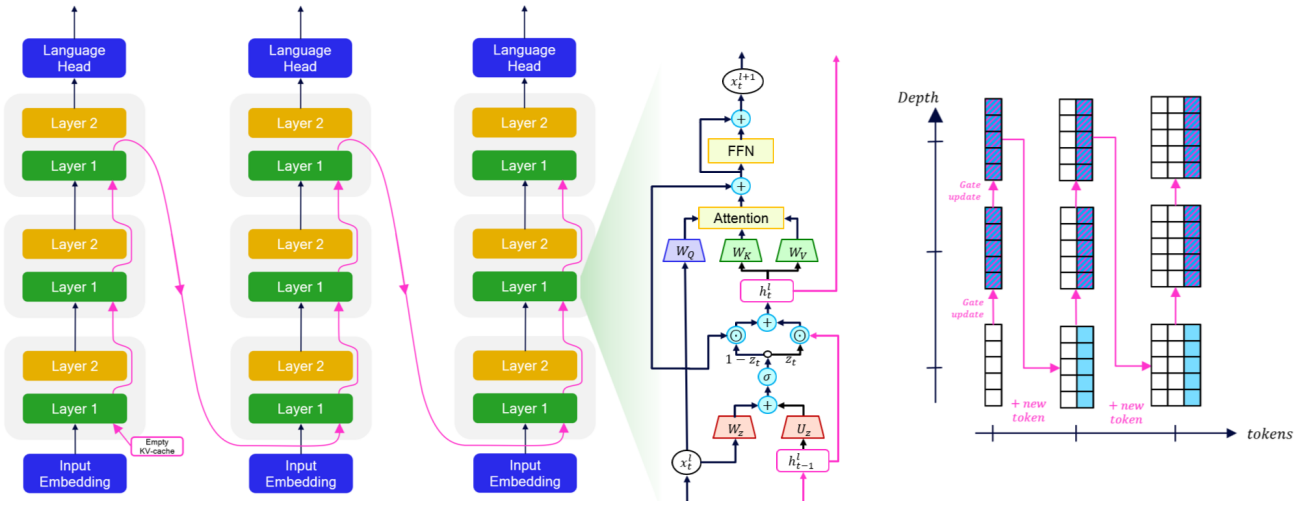
2. Related work

This section provides a concise overview of related works, see Appendix A for an extended version.

Looped transformers. While CoT (Wei et al., 2022) emphasizes horizontal reasoning, a complementary line of work explores vertical reasoning via recurrent architectures. Early approaches such as HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025), as well as adaptive-depth methods that dynamically skip or repeat layers (Li et al., 2025; Fu et al., 2026), highlight the benefits of iterative computation. More broadly, looped transformers have emerged as a strong architectural paradigm, outperforming similarly sized vanilla transformers on multi-hop reasoning, length generalization, and algorithmic tasks (Saunshi et al., 2025; Kohli et al., 2026; Fan et al., 2025; Yang et al., 2024). Despite classical optimization challenges such as instability and vanishing gradients (Dehghani et al., 2019), recent work demonstrates stable training at scale (Zhu et al., 2025; Geiping et al., 2025; Prairie et al., 2026) across different designs, including fully looped stacks and middle-cycle architectures (Geiping et al., 2025; Zeitoun et al., 2026). These results establish looped transformers as a promising direction for scalable reasoning through iterative compute.

KV cache compression and vertical sharing. Efficient KV cache management is critical in looped and long-context models, where memory typically scales with effective depth. Prior work has explored redundancy across heads, layers, and recurrence steps, including MQA/GQA for head sharing (Shazeer, 2019; Ainslie et al., 2023) and cross-layer reuse methods such as CLA and MLA (Brandon et al., 2024; DeepSeek-AI et al., 2024). In looped transformers, several approaches reduce KV growth by selectively reusing or compressing cached states, including hybrid global–local attention (Wu et al., 2025), recursion-aware caching and sharing (Bae et al., 2025), and untrained reuse across loops (Geiping et al., 2025; Zhu et al., 2025). While some of these methods can reduce memory costs in constrained settings, their effectiveness remains limited on long, complex reasoning tasks, where they often lead to performance degradation when applied to stronger models and longer reasoning traces (see Appendix B).

Training transitions and representation-level distillation. Adapting pretrained models to new architectures requires gradual transitions to avoid destabilization. Our approach is most closely related to progressive growing (Karras et al., 2018), which interpolates between existing and newly introduced components, and to subsequent work on gradual



a) Full MELT model b) Layer architecture c) KV cache updates

Figure 2. Visualization of the MELT architecture and its KV cache dynamics. The pink arrows highlight the flow of the KV cache for layer 1. **(a)** Example with 2 layers and 3 loops. As in looped transformers, layers are reused across iterations, but the KV cache is updated rather than expanded across loops. **(b)** Each layer follows a standard transformer structure, augmented with a latent state update used to compute the KV representations. **(c)** H_t^n denotes the latent state at time step t for token n in a given layer, before being projected into KV. A single shared KV cache adds one row per token and updates it across loops via the gating mechanism.

training and adaptation (Chen et al., 2026; Li & Hoiem, 2017) as well as architectural modification in LLMs (Cheng et al., 2026; Komatsuzaki et al., 2023). Complementarily, Knowledge Distillation (KD) (Hinton et al., 2015) has been used to stabilize model adaptation, with prior work showing that aligning intermediate representations improves transfer and robustness (Aguilar et al., 2020; Chen et al., 2021a). This has proven effective in LLMs, where layer-wise supervision enables compact models (Muralidharan et al., 2024) and strict activation matching mitigates representation drift in complex reasoning settings (Hao et al., 2025; Fang et al., 2026). Building on these ideas, we propose training with an *interpolated transition* and *attention-aligned distillation*.

3. Memory-Efficient Looped Transformer

3.1. Preliminaries

Notation. Throughout the paper, we use the following notation. The model has N *layers*, each a distinct transformer block with its own parameters, and uses a *hidden dimension* d for internal representations. The *sequence length*, L, corresponds to the number of tokens in the input. The *reasoning depth* or *time index*, T, refers to the number of *reasoning loops* or *time steps* applied to a single token.

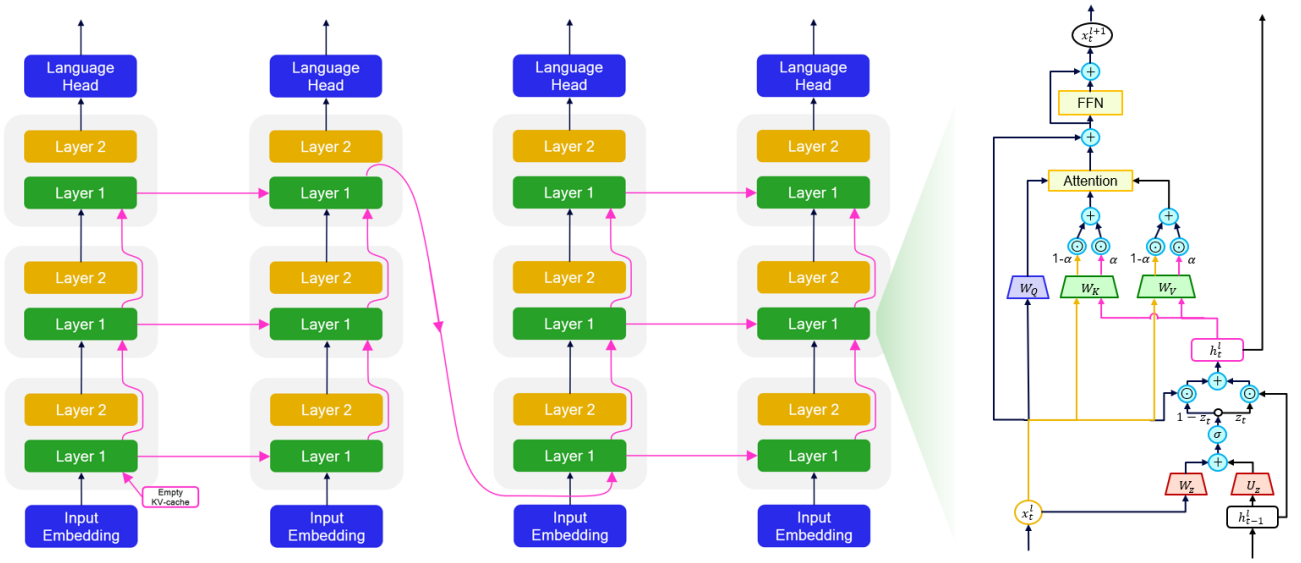
LoopLM architecture. We adopt the LoopLM architecture (Zhu et al., 2025) for causal sequence modeling, following the formulation used in prior looped-reasoning models. This design increases per-token computation without expanding the parameter count. Let $\mathrm{emb}(\cdot) : \mathbb{R}^{|V|} \to \mathbb{R}^d$ denote the token embedding map, $\mathcal{T}_\theta(\cdot) : \mathbb{R}^{L\times d} \to \mathbb{R}^{L\times d}$ a causal Transformer layer with parameters θ, and $\mathrm{lmhead}(\cdot) : \mathbb{R}^d \to \mathbb{R}^{|V|}$ the output projection. A standard (non-looped) language model composes N layers as $\mathcal{M} = \mathcal{T}_{\theta_N} \circ \cdots \circ \mathcal{T}_{\theta_1}$. In the looped setting, this stack is applied repeatedly for T iterations, so the forward pass becomes:

$$\mathrm{lmhead} \circ \underbrace{\mathcal{M} \circ \cdots \circ \mathcal{M}}_{T \text{ iterations}} \circ \mathrm{emb}(\cdot).$$

3.2. Architecture

There are three key differences that separate our architecture from LoopLM:

- The per-layer KV cache has a fixed size independent of the reasoning depth. Consequently, the total cache scales as $\mathcal{M}_{\text{MELT}} \propto \mathcal{O}(N \times L)$, compared to $\mathcal{M}_{\text{LoopLM}} \propto \mathcal{O}(N \times L \times T)$.
- Instead of appending a new state at every loop step, each loop *updates* the cached state of the token. A new state is added only at the first time step, and after all iterations these updated states are passed to subsequent tokens.
- Our gating mechanism enables each token, at each time step, to attend to keys and values that integrate information across *all* time steps of preceding tokens, rather than only the current step.



a) Chunk-wise training b) Interpolated transition

Figure 3. Visualization of the Phase 1 training techniques proposed. **(a)** Example with sequence length 4 and chunk size 2. MELT's KV cache is computed in parallel within each chunk and sequentially across chunks, balancing training efficiency with a closer approximation to autoregressive inference. **(b)** During early training steps, two KV caches are computed: the standard LoopLM version (orange) and the MELT variant (pink). These are linearly combined using a coefficient α, which increases from 0 to 1, enabling a smooth transition between the two behaviors.

Preserving query-key alignment and memory dynamics. A key design choice in MELT is to maintain a separate latent state h that evolves across iterations, from which keys and values are derived through learned projections (W_K, W_V), rather than directly updating the KV cache at each loop step. This choice is motivated by preserving semantic integrity, decoupling memory updates from attention retrieval, and maintaining query–key alignment. By evolving a latent state and projecting it into K, V space, we preserve alignment across recurrent updates while separating memory dynamics from retrieval.

This design also leads to a fundamentally different memory behavior. Standard looped transformers follow an append-only strategy, where the per-layer KV cache grows linearly with both sequence length L and reasoning depth T, i.e., $\mathcal{M}^{(l)}_{\text{LoopLM}} \propto \mathcal{O}(L \times T)$, resulting in prohibitively large memory overhead for deep reasoning. In contrast, MELT maintains a latent state $h_t^{(l)}$ with size independent from depth, yielding $\mathcal{M}^{(l)}_{\text{MELT}} \propto \mathcal{O}(L)$. The latent state is updated via a learnable gated momentum mechanism:

$$z_t^{(l)} = \sigma\left(x_t^{(l)} W_z^{(l)} + h_{t-1}^{(l)} U_z^{(l)} + b_z^{(l)}\right),$$
$$h_t^{(l)} = z_t^{(l)} \odot h_{t-1}^{(l)} + (1 - z_t^{(l)}) \odot x_t^{(l)}$$

where $x_t^{(l)}$ is the hidden state and $z_t^{(l)}$ the gating function. This reduces the depth-wise memory complexity to $\mathcal{O}(1)$ per layer, effectively recovering the footprint of non-looped transformers. As a result, the burden of retaining information shifts from explicit storage (KV cache) to the learned gating dynamics, which determine what information is preserved or overwritten over time.

Integration into the Transformer. This latent state is then used to generate the key and value representations for the current token,

$$k_t^{(l)} = h_t^{(l)} W_K^{(l)} \qquad v_t^{(l)} = h_t^{(l)} W_V^{(l)}$$

where $W_K^{(l)}, W_V^{(l)} \in \mathbb{R}^{d \times d}$ are learned projection matrices. The resulting $k_t^{(l)}$ and $v_t^{(l)}$ are appended to the KV-cache produced by earlier tokens at the same layer,

$$K_t^{(l)} = \left[K^{(l)}, k_t^{(l)}\right], \qquad V_t^{(l)} = \left[V^{(l)}, v_t^{(l)}\right],$$

which is then consumed by the attention mechanism to compute the updated hidden state $x_t^{(l+1)}$

$$x_{attn}^{(l)} = Attn^{(l)}(q^{(l)}, K_t^{(l)}, V_t^{(l)}) + x_t^{(l)}$$
$$x_t^{(l+1)} = FFN^{(l)}(x_{attn}^{(l)}) + x_{attn}^{(l)}$$

An overview of the MELT architecture is shown in Figure 2. Further theoretical analysis and analysis of gradient flow and stability is provided in Appendix E.

3.3. Training details

Chunk-wise training. A key challenge in training MELT arises from its KV-cache computation, which introduces a sequential dependency across tokens: the KV cache for token $t{+}1$ can only be computed after completing the forward pass for token t. This contrasts with standard transformers (and Ouro), where KV caches depend only on per-layer activations, enabling parallel token processing during SFT. While fully autoregressive training would respect this dependency, it is prohibitively slow, whereas bypassing the final reasoning loop restores parallelism but introduces a mismatch with inference dynamics.

To balance efficiency and fidelity, we propose **chunk-wise training**, illustrated in Figure 3. Sequences are split into fixed-length chunks processed sequentially, while computations within each chunk are performed in parallel using the current loop's latent state. Across chunks, the full computation is completed and the final latent state is propagated, better approximating autoregressive inference. The chunk size controls the fidelity–efficiency trade-off: smaller chunks more closely match inference at the cost of throughput, while larger chunks improve efficiency but introduce greater deviation.

Interpolated transition. Because chunk-wise training increases training time, we fine-tune MELT from a pretrained LoopLM rather than training from scratch, reusing the base model's acquired knowledge. However, the architectural changes introduced by MELT significantly disrupt this initialization: the model initially behaves like an untrained network and, despite fast optimization, it remains far from the original LoopLM.

To mitigate this effect and ensure a smoother transition, we introduce **training with interpolated transition**, illustrated in Figure 3. During training, two KV pairs are computed in parallel: KV_{base} from the hidden states as in a standard LoopLM and KV_{MELT} from the MELT architecture. The KV values used by the model is a linear interpolation

$$KV = \alpha\, KV_{\text{MELT}} + (1-\alpha)\, KV_{\text{base}},$$

where α increases linearly from 0 to 1 during training, enabling a smooth transition from LoopLM to MELT.

To further preserve alignment with the pretrained model, we apply Knowledge Distillation (Hinton et al., 2015) using the initial LoopLM as teacher, applying supervision at all reasoning loops. This denser signal improves convergence and stabilizes training.

Attention-aligned distillation. After the interpolation phase reaches $\alpha = 1$, the model operates entirely under MELT dynamics. While training could simply continue from this point, we observe that unconstrained continuation degrades performance, suggesting that MELT representations drift away from the pretrained LoopLM behavior.

To prevent this, we introduce a second training phase. In this phase, the original LoopLM is kept frozen and used as a teacher for knowledge distillation, complemented by an attention-alignment loss that aligns MELT's post-attention representations with those of the teacher at every layer and loop (see Figure 5). The resulting objective is

$$\mathcal{L} = \mathcal{L}_{\text{KD}} + \beta \frac{1}{NT} \sum_{l=1}^{N} \sum_{t=1}^{T} \left\| o_{\text{MELT}}^{(l,t)} - \text{sg}\left(o_{\text{LoopLM}}^{(l,t)} \right) \right\|_2^2,$$

where $o_{\text{MELT}}^{(l,t)}$ and $o_{\text{LoopLM}}^{(l,t)}$ denote the post-attention representations at layer l and loop t, β controls the strength of the alignment term, and $\text{sg}(\cdot)$ denotes the stop-gradient operator. This term enforces alignment at all layers and loops, stabilizing training and further reducing the gap to the original LoopLM (see Table 4).

4. Experimental results

4.1. Experimental setup

We initialize our model, MELT-1.6B, using the pretrained weights of Ouro-1.4B-Thinking (Zhu et al., 2025), except for the new gating parameters, which are initialized randomly. Because MELT modifies the KV cache structure and introduces randomly initialized gating parameters, this hybrid initialization leads to initially incoherent outputs. To address this, we fine-tune the full model in two stages, as described in Subsection 3.3. In the first stage, we use chunk-wise and interpolating training and, in the second stage, we apply chunk-wise training with Attention-Aligned Distillation. Both training on AceReason-1.1-SFT (Liu et al., 2025) and OpenThoughts3 (Guha et al., 2025) datasets, focused on mathematical reasoning and coding. A summary of all training hyperparameters used in this stage is shown in Table 6. In total, training required 130 hours on a node with 8 H100 GPUs (80GB), corresponding to 1,040 GPU-hours. Further details on the compute used for preliminary experiments, ablations, and testing are provided in Appendix D.

To evaluate the reasoning capabilities of MELT, we benchmark the model on six mathematical reasoning benchmarks (AIME24 (Mathematical Association of America, 2024), AIME25 (Mathematical Association of America, 2025), AIME26 (Mathematical Association of America, 2026), AMC23 (Mathematical Association of America, 2023), MATH500 (Lightman et al., 2023), OlympiadBench (He et al., 2024)) and four general reasoning benchmarks (GPQA (Rein et al., 2023), HLE (Phan et al., 2026), MMLU-Red (Gema et al., 2024; Hendrycks et al., 2021b), Humaneval (Chen et al., 2021b)). For context,

Table 1. Performance comparison across benchmarks. We use **bold** and underlining to denote the best and second-best performance, respectively.

Dataset	Metric	Ouro-1.4B Thinking	MELT 1.6B	Qwen3 1.7B	Gemma4 E2B	Qwen3.5 2B	DeepSeek-R1 1.5B
AIME24	pass@1	**50.2**±1.6	46.7±1.6	43.1±1.5	40.6±1.8	19.0±1.3	32.1±1.6
	pass@10	**81.5**±1.9	79.9±2.4	76.2±2.6	68.5±2.8	47.0±2.5	75.7±3.5
AIME25	pass@1	**36.7**±1.5	33.3±1.3	33.1±1.3	26.5±1.3	16.9±1.1	20.4±1.2
	pass@10	**69.0**±2.5	61.9±2.7	58.6±2.7	50.1±2.9	37.1±2.8	46.0±2.7
AIME26	pass@1	**44.0**±1.5	41.0±1.6	31.7±1.4	36.0±1.7	16.0±1.3	19.8±1.3
	pass@10	73.2±2.4	**75.5**±2.0	61.5±2.9	58.3±2.6	46.7±2.4	48.7±2.7
AMC23	pass@1	81.2±1.2	80.2±1.2	79.2±1.2	**82.7**±1.1	64.4±1.4	70.9±1.3
	pass@10	96.6±1.0	**97.8**±1.5	**97.8**±1.5	95.0±1.9	92.0±1.2	96.1±1.4
MATH-500	accuracy	**94.4**±1.0	93.4±1.1	90.6±1.3	87.6±1.5	79.4±1.8	84.2±1.6
OlympiadB	accuracy	**67.5**±1.9	64.7±2.0	63.5±2.0	62.7±2.0	48.4±2.1	54.2±2.1
Avg math	pass@1	**62.3**	59.9	56.9	56.0	40.7	46.9
GPQA	accuracy	40.8±2.3	42.6±2.3	37.3±2.3	39.1±2.3	**45.1**±2.4	31.9±2.2
HLE	accuracy	**2.7**±0.9	2.0±0.8	1.3±0.7	2.0±0.8	1.7±0.7	2.0±0.8
MMLU-Red	accuracy	74.2±0.6	74.2±0.6	73.8±0.6	**75.3**±0.6	**75.3**±0.6	53.3±0.7
Humaneval	accuracy	76.8±3.3	**81.7**±3.0	71.3±3.5	61.6±3.6	26.2±3.4	57.3±3.9
Avg non-math	pass@1	48.6	**50.1**	45.9	45.5	37.1	36.1

we compare its performance with the state-of-the-art non-looped models of its size (Qwen3-1.7B (Yang et al., 2025), Gemma4-E2B (Google, 2024), Qwen3.5-2B (Team, 2026), DeepSeek-R1-1.5B (Guo et al., 2025)), as well as the looped model Ouro-1.4B-Thinking (Zhu et al., 2025), from which MELT-1.6B is derived. We evaluate all models with LightEval v0.8.1, using the default benchmark prompts, extraction procedures, and evaluation settings. Following (Zhu et al., 2025), we use temperature 1.0 and top-p 0.7; all evaluations use a maximum completion length of 32k tokens.

4.2. Results

Table 1 shows that MELT consistently outperforms all non-looped baselines across both mathematical and general reasoning benchmarks, while maintaining a comparable memory footprint. In particular, MELT achieves superior performance on AIME24, AIME26, MATH500, OlympiadBench, MMLU, and HumanEval. It is only surpassed by Qwen3-1.7B on AIME25 and AMC23, and by Gemma4-E2B on GPQA. Overall, these results demonstrate that MELT performs strongly across both mathematical and general reasoning tasks.

Compared to Ouro, MELT is slightly behind across most benchmarks, which is expected given that Ouro retains a full per-loop KV cache and thus benefits from substantially higher memory usage. Interestingly, however, MELT outperforms Ouro on HumanEval. We discuss slight discrepancies with the Ouro paper benchmarks in Appendix F. Overall, these results highlight that MELT achieves a strong performance–efficiency trade-off, delivering superior results to non-looped models while approaching the performance of memory-intensive looped architectures.

4.3. Exact memory usage

In this subsection we report *exact* KV-cache memory usage numbers extracted from `vLLM` (Kwon et al., 2023), and we combine them with a simple weight-memory estimate to obtain an end-to-end VRAM requirement for long generations (32k tokens). This analysis highlights the substantial improvements achieved by MELT compared to Ouro, since for long-context generation the dominant contributor to memory usage is the KV-cache. For each model, we report:

KV-cache per token (MB/token): obtained directly from *vLLM*'s reported metrics. *Model memory (GB)*: the memory required to store the model weights, obtained as $M_{\text{model}} = 2 \cdot \#\text{params}$ bytes. *KV-cache for a 32k-token generation (GB)*: the total memory consumed by the KV-cache when generating a 32,768-token sequence, computed as $M_{\text{KV},32k} = 32768 \cdot M_{\text{KV/token}}$. *Total memory for a 32k generation (GB)*:

Table 2. Exact KV-cache memory (from `vLLM`) and derived VRAM requirements for generating a 32k-token sequence.

Model	KV-cache (MB/token)	Model memory (GB)	KV-cache for 32k (GB)	Total for 32k (GB)
MELT-1.6B	0.196608	3.272	6.29	9.49
Ouro-1.4B-Thinking	0.786432	2.869	25.17	27.97
Memory improvement	×4	–	×4	×2.95
Qwen3-1.7B	0.114688	3.442	3.67	7.07

the sum of model model memory and KV-cache for a 32k-token generation.

As shown in Table 2, Ouro exhibits the largest KV-cache footprint, as its loop-specific KV growth causes memory to scale linearly with the number of reasoning loops. In contrast, MELT decouples reasoning depth from KV growth by maintaining a constant-size latent state instead of appending new KV entries, reducing memory by $\sim$ 3-4×. Although Qwen remains slightly more memory efficient in KV usage, the gap is small: for a 32k-token generation, Ouro exceeds Qwen by $\sim$ 20 GB, while MELT is only $\sim$ 2.5 GB higher. This difference stems from Qwen's use of Multi-Query Attention (MQA), which reduces KV memory by sharing keys and values across query heads, whereas MELT does not employ MQA.

4.4. Ablation studies

4.4.1. Gate mechanism variants

A core component of MELT is its gated update mechanism, which controls how loop-specific information is accumulated into the latent state. To assess the necessity and effectiveness of this design, we train a set of variants in which the proposed element-wise gating mechanism is replaced with simpler aggregation schemes. All other components are kept identical, and we restrict training to the first stage to ensure a controlled comparison. Concretely, we compare the full MELT model against the following variants:

Mean: the KV cache is computed as the average of the KV representations produced by all loops up to the current step. *EMA-0.2*: the KV cache is computed as an exponential moving average (EMA) of the KV representations up to the current step. The chosen decay factor (0.2) matches the average gate value observed in our trained MELT models. This is equivalent to the gated mechanism with gate value fixed to 0.2. *Last*: the KV cache is constructed solely from the final reasoning loop, discarding information from earlier loops. *Single-gated*: the gated update is replaced with a scalar gate per token, such that a single gating value modulates the entire hidden state uniformly, rather than using an element-wise gate.

Table 3 shows that MELT (element-wise gating, after Phase 1 training) consistently achieves the best performance. Among variants without additional parameters, *Last* performs best and is comparable to *Single-gated*, highlighting the importance of selective aggregation and the more effective utilization of information from later reasoning loops. We evaluate all ablations after Phase 1 training to isolate the effect of the gating mechanism.

Table 3. MELT's gating mechanism ablation after Phase 1 of training. **Bold** denotes best performance.

	AIME24	AIME25	AMC23	MATH-500
MELT-1.6B-P1	**44.8**	**32.9**	**77.7**	**92.8**
Mean	29.0	23.3	68.8	83.2
EMA-0.2	30.2	21.5	68.6	84.6
Last	33.7	24.0	69.7	84.0
Single-gated	34.4	23.1	66.9	85.6

4.4.2. Component removal

We next perform a *component removal ablation* to assess the importance of individual training components in MELT. Starting from the full model, we progressively remove elements of the training procedure one by one, following their order of introduction, and fully retrain the model after each removal to ensure a fair comparison.

Specifically, we remove training mechanisms one by one to isolate their contribution, following the sequence: (i) removing attention-aligned distillation, using only the first training phase, (ii) additionally removing interpolation training, reverting to a direct transition from LoopLM to MELT; (iii) removing knowledge distillation on all loops, reducing training to standard SFT; and (iv) replacing chunk-wise training with fully parallel SFT.

As shown by Table 4, each component yields a clear and consistent improvement over the preceding configuration across all benchmarks. Removing attention-aligned distillation (Phase 2) already causes a notable performance drop in most benchmarks, demonstrating its critical role in consolidating the learned MELT representations. Further removing interpolated transition within Phase 1 degrades performance, confirming that a smooth LoopLM-to-MELT transition is

Table 4. Component removal ablation for MELT. Starting from the full two-phase training recipe, components are progressively removed one by one (top to bottom). **Bold** denotes best performance.

	AIME24	AIME25	AMC23	MATH-500
MELT-1.6B	**46.7**	**33.3**	**80.2**	**93.4**
− Att-aligned dist.	44.8	32.9	77.7	92.8
− Interpolated trans.	35.4	26.9	73.0	86.6
− GKD-allLoops	35.8	24.4	67.2	85.2
− Chunk-wise train	0.0	0.0	0.0	0.0

essential. Disabling knowledge distillation worsens results further, and eliminating chunk-wise training leads to complete failure, confirming that respecting MELT's sequential KV dynamics during training is indispensable. Overall, these results demonstrate that MELT's performance arises from the cumulative effect of its two-phase training components, rather than from any single component in isolation.

5. Limitations and future work

A limitation inherited from Ouro is that the number of recurrent loops is fixed at inference time. While this provides a simple mechanism to control compute, it does not account for the fact that different inputs and tokens may require varying amounts of reasoning. Notably, MELT's constant-size latent state makes it particularly well-suited for future extensions with adaptive loop depth, enabling dynamic allocation of reasoning steps based on input complexity.

Another limitation carried over from Ouro is that our current implementation does not yet explore MQA. Extending MELT to multi-query attention remains an important direction, as MQA can reduce memory bandwidth and KV cache overhead during inference. Thanks to MELT's constant-memory design, combining it with MQA is especially promising and could further improve efficiency, potentially closing the remaining gap in memory usage with standard Transformer baselines.

Finally, MELT requires sequential KV updates during training, which constrains parallelism compared to standard transformer training. While our chunk-wise training and distillation procedure provides a practical adaptation path, developing more parallelizable training strategies remains an important direction for scaling MELT to larger models, longer reasoning horizons, and broader application domains.

6. Conclusion

We introduce MELT, an architecture that enables deep latent reasoning in looped transformers by decoupling memory usage from reasoning depth. By replacing the append-only KV-cache with a **gated, constant-size latent state**, MELT allows inference-time compute to scale without incurring linear memory growth. When integrated into Ouro, it achieves competitive reasoning performance relative to similarly sized baselines.

Training MELT, however, requires particular care. Its KV cache computation introduces a sequential dependency across tokens, preventing the fully parallel token processing typically used during training. We address this challenge through **chunk-wise training**, which provides a controllable trade-off between inference fidelity (smaller chunks) and training efficiency (larger chunks). In addition, directly reusing Ouro's architecture and weights proves challenging due to the substantial architectural changes introduced by MELT. To mitigate this, we employ two complementary techniques on top of KD: an **interpolated transition** that enables a smooth shift from LoopLM to MELT dynamics, followed by **attention-aligned distillation**, where a frozen LoopLM teacher provides layer-wise supervision to stabilize and consolidate the learned representations.

Empirically, MELT delivers strong and consistent performance across both mathematical and general reasoning benchmarks. Notably, it surpasses similarly sized standard Transformer baselines while operating under the same constant memory budget, demonstrating that improved reasoning capability can be achieved without increasing memory. To our knowledge, MELT is the first architecture to exceed the performance of standard models with the same memory footprint. These results highlight the effectiveness of looped architectures and demonstrating that iterative computation provides meaningful gains in reasoning even when memory is strictly constrained.

Impact Statement

This paper introduces a potential new approach for advancing large language models by improving their efficiency and reasoning capabilities. As with most progress in this domain, the societal impact of this work is largely tied to the broader evolution of LLM technologies rather than to any uniquely differentiated application or risk. Improvements in model efficiency and reasoning may contribute to more capable AI systems across a range of downstream uses, including scientific research, software development, education, and general-purpose assistants. At the same time, these advances may indirectly amplify both the benefits (e.g., increased productivity, accessibility of knowledge) and risks (e.g., misuse, over-reliance, or propagation of errors) already associated with LLMs.

References

Aguilar, G., Ling, Y., Zhang, Y., Yao, B., Fan, X., and Guo, C. Knowledge Distillation from Internal Representations, Jan-

uary 2020. URL http://arxiv.org/abs/1910.03723. arXiv:1910.03723 [cs].

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models. *arXiv preprint arXiv:2305.13245*, 2023.

Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., Fisch, A., Harutyunyan, H., Ji, Z., Courville, A., and Yun, S.-Y. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Blayney, H., Álvaro Arroyo, Obando-Ceron, J., Castro, P. S., Courville, A., Bronstein, M. M., and Dong, X. A mechanistic analysis of looped reasoning language models, 2026. URL https://arxiv.org/abs/2604.11791.

Brandon, W., Mishra, M., Nrusimha, A., Panda, R., and Kelly, J. R. Reducing transformer key-value cache size with cross-layer attention, 2024. URL https://arxiv.org/abs/2405.12981.

Chen, D., Mei, J.-P., Zhang, Y., Wang, C., Feng, Y., and Chen, C. Cross-Layer Distillation with Semantic Calibration, August 2021a. URL http://arxiv.org/abs/2012.03236. arXiv:2012.03236 [cs].

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021b.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021c. URL https://arxiv.org/abs/2107.03374.

Chen, T., Xu, X., Yin, L., Chen, H., Wang, Y., Diao, S., and Yang, C. Progressive residual warmup for language model pretraining, 2026. URL https://arxiv.org/abs/2603.05369.

Cheng, Z., Yang, H.-B., Huang, W.-Y., and Li, J.-L. Attention editing: A versatile framework for cross-architecture attention conversion, 2026. URL https://arxiv.org/abs/2604.05688.

DeepSeek-AI, Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Yang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Chen, J., Yuan, J., Qiu, J., Song, J., Dong, K., Gao, K., Guan, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Zhang, L., Li, M., Wang, M., Zhang, M., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019.

Fan, Y., Du, Y., Ramchandran, K., and Lee, K. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2025. doi: 10.48550/ARXIV.2409.15647. URL https://arxiv.org/abs/2409.15647. ICLR 2025.

Fang, L., Yu, X., Cai, J., Chen, Y., Wu, S., Liu, Z., Yang, Z., Lu, H., Gong, X., Liu, Y., Ma, T., Ruan, W., Abbasi, A., Zhang, J., Wang, T., Latif, E., You, W., Jiang, H., Liu, W., Zhang, W., Kolouri, S., Zhai, X., Zhu, D., Zhong, W., Liu, T., and Ma, P. Knowledge Distillation and Dataset Distillation of Large Language Models: Emerging Trends, Challenges, and Future Directions, January 2026. URL http://arxiv.org/abs/2504.14772. arXiv:2504.14772 [cs].

Fu, T., You, Y., Chen, Z., Dai, G., Yang, H., and Wang, Y. Think-at-hard: Selective latent iterations to improve reasoning language models, 2026. URL https://arxiv.org/abs/2511.08577.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025. doi: 10.48550/ARXIV.2502.05171. URL https://arxiv.org/abs/2502.05171.

Gema, A. P., Leang, J. O. J., Hong, G., Devoto, A., Mancino, A. C. M., Saxena, R., He, X., Zhao, Y., Du, X., Madani, M. R. G., Barale, C., McHardy, R., Harris, J., Kaddour, J., van Krieken, E., and Minervini, P. Are we done with mmlu?, 2024.

Google. Gemma open models, 2024. URL https://ai.google.dev/gemma.

Guha, E., Marten, R., Keh, S., Raoof, N., Smyrnis, G., Bansal, H., Nezhurina, M., Mercat, J., Vu, T., Sprague, Z., Suvarna, A., Feuer, B., Chen, L., Khan, Z., Frankel, E., Grover, S., Choi, C., Muennighoff, N., Su, S., Zhao, W., Yang, J., Pimpalgaonkar, S., Sharma, K., Ji, C. C.-J., Deng, Y., Pratt, S., Ramanujan, V., Saad-Falcon, J., Li, J., Dave, A., Albalak, A., Arora, K., Wulfe, B., Hegde, C., Durrett, G., Oh, S., Bansal, M., Gabriel, S., Grover, A., Chang, K.-W., Shankar, V., Gokaslan, A., Merrill, M. A., Hashimoto, T., Choi, Y., Jitsev, J., Heckel, R., Sathiamoorthy, M., Dimakis, A. G., and Schmidt, L. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Habib, N., Fourrier, C., Kydlíček, H., Wolf, T., and Tunstall, L. Lighteval: A lightweight framework for llm evaluation, 2023. URL https://github.com/huggingface/lighteval.

Hao, J., Huang, Q., Liu, H., Xiao, X., Ren, Z., and Yu, J. A Token is Worth over 1,000 Tokens: Efficient Knowledge Distillation through Low-Rank Clone, December 2025. URL http://arxiv.org/abs/2505.12781. arXiv:2505.12781 [cs].

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z. L., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., Zhou, J., Hou, L., Li, J., and Sun, M. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14138–14166, 2024. URL https://aclanthology.org/2024.acl-long.762.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring Massive Multitask Language Understanding, January 2021b. URL http://arxiv.org/abs/2009.03300. arXiv:2009.03300 [cs].

Hinton, G., Vinyals, O., and Dean, J. Distilling the Knowledge in a Neural Network, March 2015. URL http://arxiv.org/abs/1503.02531. arXiv:1503.02531 [stat].

Jolicoeur-Martineau, A. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation, 2018. URL https://arxiv.org/abs/1710.10196.

Kohli, H., Parthasarathy, S., Sun, H., and Yao, Y. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers, 2026. URL https://arxiv.org/abs/2604.07822.

Komatsuzaki, A., Puigcerver, J., Lee-Thorp, J., Ruiz, C. R., Mustafa, B., Ainslie, J., Tay, Y., Dehghani, M., and Houlsby, N. Sparse upcycling: Training mixture-of-experts from dense checkpoints, 2023. URL https://arxiv.org/abs/2212.05055.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Li, Z. and Hoiem, D. Learning without forgetting, 2017. URL https://arxiv.org/abs/1606.09282.

Li, Z., Li, Y., and Zhou, T. Skip a layer or loop it? test-time depth adaptation of pretrained llms. *arXiv preprint arXiv:2507.07996*, 2025. doi: 10.48550/ARXIV.2507.07996. URL https://arxiv.org/abs/2507.07996.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

Liu, Z., Yang, Z., Chen, Y., Lee, C., Shoeybi, M., Catanzaro, B., and Ping, W. AceReason-Nemotron 1.1: Advancing Math and Code Reasoning through SFT and RL Synergy, June 2025. URL http://arxiv.org/abs/2506.13284.

Mathematical Association of America. American mathematics competitions (amc) 10/12 2023, 2023. URL https://maa.org/.

Mathematical Association of America. American invitational mathematics examination (aime) 2024, 2024. URL https://maa.org/. Problems I and II.

Mathematical Association of America. American invitational mathematics examination (aime) 2025, 2025. URL https://maa.org/. Problems I and II.

Mathematical Association of America. American invitational mathematics examination (aime) 2026, 2026. URL https://maa.org/. Problems I and II.

Muralidharan, S., Sreenivas, S. T., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. Compact Language Models via Pruning and Knowledge Distillation, November 2024. URL http://arxiv.org/abs/2407.14679. arXiv:2407.14679 [cs].

Phan, L., Gatti, A., Li, N., Khoja, A., Kim, R., Ren, R., Hausenloy, J., Zhang, O., Mazeika, M., Hendrycks, D., Han, Z., Hu, J., Zhang, H., Zhang, C. B. C., Shaaban, M., Ling, J., Shi, S., Choi, M., Agrawal, A., Chopra, A., Nattanmai, A., McKellips, G., Cheraku, A., Suhail, A., Luo, et al. A benchmark of expert-level academic questions to assess ai capabilities. *Nature*, 649(8099): 1139–1146, January 2026. ISSN 1476-4687. doi: 10.1038/s41586-025-09962-4. URL http://dx.doi.org/10.1038/s41586-025-09962-4.

Prairie, H., Novack, Z., Berg-Kirkpatrick, T., and Fu, D. Y. Parcae: Scaling laws for stable looped language models, 2026. URL https://arxiv.org/abs/2604.12946.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025. doi: 10.48550/ARXIV.2502.17416. URL https://arxiv.org/abs/2502.17416. ICLR 2025.

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

Team, Q. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., Rasul, K., and Gallouédec, Q. TRL: Transformers Reinforcement Learning, 2020. URL https://github.com/huggingface/trl.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. Hierarchical Reasoning Model, 2025. URL https://arxiv.org/abs/2506.21734. Version Number: 3.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Wu, B., Chen, M., Luo, X., Yan, S., Yu, Q., Xia, F., Zhang, T., Zhan, H., Zhong, Z., Zhou, X., Qiao, S., and Bin, X. Parallel loop transformer for efficient test-time computation scaling, 2025. URL https://arxiv.org/abs/2510.24824.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., and other. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2024. doi: 10.48550/ARXIV.2311.12424. URL https://arxiv.org/abs/2311.12424. Accepted at ICLR 2024.

Zeitoun, A., Torroba-Hennigen, L., and Kim, Y. Hyperloop transformers, 2026. URL https://arxiv.org/abs/2604.21254.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., Li, L., Shi, J., Ma, K., Li, S., Kergan, T., Smith, A., Qu, X., Hui, M., Wu, B., Min, Q., Huang, H., Zhou, X., Ye, W., Liu, J., Yang, J., Shi, Y., Lin, C., Zhao, E., Cai, T., Zhang, G., Huang, W., Bengio, Y., and Eshraghian, J. Scaling latent reasoning via looped language models, 2025. URL https://arxiv.org/abs/2510.25741.

A. Extended Related Work

Looped transformers. While CoT (Wei et al., 2022) and other ITC techniques have recently been highly influential, a complementary direction has emerged that focuses on vertical reasoning via recurrent architectures. Simple recurrent architectures such as HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025) have demonstrated strong performance on targeted reasoning tasks, while transformer-based models have also been modified to incorporate looping mechanisms. For instance, Li et al. (2025) propose a per-sample test-time depth adaptation, where a pretrained LLM's layers are treated as modules that can be skipped or repeated (looped) and reordered to form a sample-specific chain-of-layers. On a similar direction, Fu et al. (2026) introduced adaptive computation budgets, employing a classifier to dynamically allocate additional latent iterations for difficult tokens that can be skipped or repeated (looped) and reordered to form a sample-specific chain-of-layers.

Most notably within this line of work, looped transformers have emerged as a powerful architectural choice. Existing studies on simplified setups have shown that, compared to similar-sized vanilla transformers, looped transformers show superior capacity at multi-hop reasoning (Saunshi et al., 2025; Kohli et al., 2026), lenght generalization (Fan et al., 2025; Kohli et al., 2026) and learning algorithms (Yang et al., 2024). Despite the challenges of scaling such architectures when unrolled over many steps, most notably optimization instability and vanishing gradients (Dehghani et al., 2019), recent studies demonstrate that looped transformers can already be trained stably at the scale of several billion parameters (Zhu et al., 2025; Geiping et al., 2025), including reasoning-focused models (Zhu et al., 2025). Existing approaches generally follow two strategies: either looping transformer layers directly across iterations (Zhu et al., 2025), or looping only a central subset of layers using a middle-cycle design with fixed prelude and coda blocks (Geiping et al., 2025; Zeitoun et al., 2026). Additional work has reported favorable scaling behavior (Prairie et al., 2026), proposed architectural refinements (Zeitoun et al., 2026), and provided mechanistic insights into the internal dynamics of looped models (Blayney et al., 2026), further supporting looped transformers as a robust and promising research direction.

KV cache compression and vertical sharing. Efficient KV cache management is central in scaling recurrent models and long-context regimes, where cache cost scales with effective depth. Beyond head-level sharing via MQA (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023), recent work exploits vertical redundancy across layers or recurrence steps. Cross-Layer Attention (CLA) (Brandon et al., 2024) shows that KV representations remain relatively stable across adjacent layers, allowing multiple layers to read from a shared cache. DeepSeek-AI et al. (2024) further compresses this information via Multi-Layer Attention (MLA) using low-rank projections.

More specifically, there are some works that address the growing KV Cache issue in looped transformers. Wu et al. (2025) propose a KV-cache computation method that combines a global component from the first loop with a local component attending to a sliding window of recent tokens in the current loop. Bae et al. (2025) propose two key mechanisms for KV cache efficiency: Recursion-wise Caching, which selectively updates and attends only to the KV pairs of active "thinking" tokens at each depth, and Recursive KV Sharing, which reuses the initial cache from the first loop across all subsequent steps. Geiping et al. (2025) show that the looped transformer, without additional training, naturally reuses the first loop KV-cache entry from previous tokens, independent of when recurrence stops, and can further compress the cache by sharing entries periodically across recurrent steps. Finally, Zhu et al. (2025) show that, without additional training, retaining only the last or average KV-cache and reusing it across all loops can preserve performance on some tasks. Although some of these approaches achieve no or moderate performance drops in their respective settings, their applicability to practical long-reasoning remains unclear as their evaluations are mostly limited to weaker looped models and short or constrained generation settings. In fact, our analysis (Appendix B) suggests that directly applying untrained cache-sharing methods to Ouro, including those proposed by Geiping et al. (2025) and Zhu et al. (2025), significantly degrades performance on long reasoning tasks.

Interpolated transition mechanism. Our interpolated transition mechanism is most closely related to the progressive growing strategy of Karras et al. (2018). They progressively grow the generator and discriminator from low to high resolutions, using a linearly increasing parameter α to smoothly interpolate between the old lower-resolution pathway and the newly added higher-resolution pathway, thereby avoiding abrupt shocks to previously trained layers. Later work has explored related forms of gradual training (Chen et al., 2026), task adaptation (Li & Hoiem, 2017), and architecture expansion, including recent approaches for modifying LLM architectures (Cheng et al., 2026; Komatsuzaki et al., 2023). However, these methods typically rely on distillation objectives or parameter reuse rather than an explicit fade-in between two competing architectures.

Activation-level knowledge distillation. An extensive body of literature exists on KD (Hinton et al., 2015). We build on the line of work that aligns intermediate representations rather than relying solely on final output logits. Notable examples within this vast space include foundational works that distill internal states to compress structural knowledge (Aguilar et al., 2020) and cross-layer mechanisms designed to calibrate semantic alignment between teacher and student (Chen et al., 2021a). In the context of large language models, Muralidharan et al. (2024) recently demonstrated the efficacy of combining structural pruning with layer-wise KD to derive highly accurate, compact models without full retraining. Furthermore, recent frameworks emphasize strict internal activation alignment to prevent catastrophic drift (Hao et al., 2025) and preserve complex, multi-step reasoning trajectories (Fang et al., 2026). Building on these foundations, our approach addresses representation drift in continuous recurrent architectures by applying knowledge distillation across all MELT loops. We further introduce an attention-alignment loss that explicitly regularizes MELT's post-attention representations against those of a frozen LoopLM teacher at every layer and reasoning loop.

B. Analysis of existing KV-Cache sharing methods on long reasoning

As discussed in the related work, several studies on KV-cache sharing in looped models suggest that caches can be reused to reduce memory footprint with minimal performance degradation. However, their applicability to practical long-reasoning settings remains unclear, as existing evaluations are largely restricted to weaker looped models and short or otherwise constrained generation regimes. In this section, we investigate whether these methods extend to state-of-the-art looped models (Ouro (Zhu et al., 2025)) on long-horizon reasoning tasks.

To do so, we evaluate several untrained KV-cache sharing strategies in which the cache is reused across recurrent loops. Specifically, we consider sharing the KV cache from the last loop, following Zhu et al. (2025), and from the first loop, following Geiping et al. (2025). Since Zhu et al. (2025) observe that preserving the full prompt KV cache can be beneficial, we evaluate both variants for each strategy: one that keeps the original KV cache for the prompt and one that applies the first/last-loop sharing rule to the prompt as well.

All four untrained KV-sharing variants obtain zero performance on several reasoning benchmarks (full results can be seen in Table 5). This contrasts with Zhu et al. (2025), who report that last-loop KV sharing with prefill achieves performance comparable to Ouro on GSM8K and MATH-500. We hypothesize that this discrepancy is partly due to their few-shot CoT evaluation setup (Appendix C.1 in Zhu et al. (2025)), which constrains the generation format and reduces the likelihood of the model drifting during decoding. In our setting, the failure mode is apparent in qualitative outputs: as illustrated in Figure 4, generations with last-loop sharing and prefill often begin coherently but eventually degenerate during extended reasoning. This behavior is consistent with an accumulation of KV-cache mismatch errors: as generation proceeds farther from the prompt, the shared cache increasingly deviates from the cache produced by the original model. These results motivate the need for a constant-memory KV-cache method that remains stable over long reasoning traces, which is precisely the goal of our approach.

Table 5. Performance of untrained KV-cache sharing strategies for Ouro-1.4B-Thinking. We compare first-loop and last-loop KV-cache sharing, with and without preserving the original prompt KV cache.

	AIME24		**AIME25**		**AMC23**		**MATH-500**
	pass@1	pass@10	pass@1	pass@10	pass@1	pass@10	
Ouro-1.4B-Thinking	**50.2**	**81.5**	**36.7**	**69.0**	**81.2**	**96.6**	**94.4**
Last loop (w/ prefill)	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Last loop	0.0	0.0	0.0	0.0	0.0	0.0	0.0
First loop (w/ prefill)	0.0	0.0	0.0	0.0	0.0	0.0	0.0
First loop	0.0	0.0	0.0	0.0	0.0	0.0	0.0

¡think¿Okay, so I need to simplify tan 100 degrees plus 4 times sin 100 degrees. Hmm, let me think about how to approach this. I remember some trigonometric identities, maybe I can use them here.

First, let me write down the expression: tan(100°) + 4 sin(100°). Hmm, 100 degrees is in the second quadrant, so tan is negative there because tangent is sine over cosine, and cosine is negative in the second quadrant. Sin is positive. But maybe there's a way to combine these terms.

Alternatively, maybe I can use some angle addition formulas or convert tangent into sine and cosine? Let me see. Since tangent is sine over cosine, so tan(100°) = sin(100°)/cos(100°). So then the expression becomes [sin(100°)/cos(100°)] +4 sin(100°).

Hmm, so if I get a common denominator, maybe I can combine them? Let's see. The common denominator would be cos(100°). So that would be [sin(100°) +4 sin(100°) cos(100°)] divided by cos(100°). Wait, but that might not help much. Hmm.

Alternatively, maybe there is a clever way to write this as some sine or cosine function with some angle? Let me think.

Wait, 100 degrees is 80 degrees plus 20 degrees. 80 is 60 plus 20, but not sure. Maybe I can use some addition formulas?

Alternatively, perhaps there is a way to express 4 sin(100°) as some sine of another angle. Maybe with some multiple angle formulas?

Alternatively, maybe I can write 100° as 60° +40°, because h, 60+40 is h 100. Let me see. Let me think.

Wait, maybe I can use the identity for tan(A+B), but not sure. Alternatively, maybe I can write 100° as h 80°+20°, but maybe maybe I can't think of a direct identity.

Alternatively, maybe I can use the identity that tan(x) +4 sin(x) = something. Maybe there is a formula for that?

Wait, let me try tohe following:

Let me consider the expressionation: tan(100° +4 sin(100°). Maybe I can writet it as [tan(100°) +4 sin(100°)]. to see if I can findd some angle such that this expressionation is equal to some sine or cosine functiontian ofthat angle.

Hmm, maybe maybe maybe maybe maybe I know maybe that maybe the expressionation can be simplified using the identity:

tan(A) +4 sin(A)=sqrt(5) something.

Wait, I think I heard of a formula that tan((+4 sin()=sqrt(5) or something like like like. Let me check that

Let me try to see this. Let me think that if A=1 , then the expressionation would is tan()+=4 sin(), but probably that the value of thetan 100° plus h teh 4 sin100° is approximately equal to teh sqrt(5).

h, let me seech that.

h,

Hmmm, let me seeh theangleng in that.

h, maybe,

,

h,

Wait,
h, maybe,
h, h,
the expressionason tan(h +4 sin(h')= teh sqrt(5).

Maybe, (...)

Figure 4. Example reasoning trace in Ouro-1.4B-Thinking illustrating the failure mode of last-loop KV-cache sharing with prefill.

C. Attention Alignment Loss

In this section, we provide Figure 5, to support the explanation of the Attention Alignment Loss provided in Subsection 3.3.

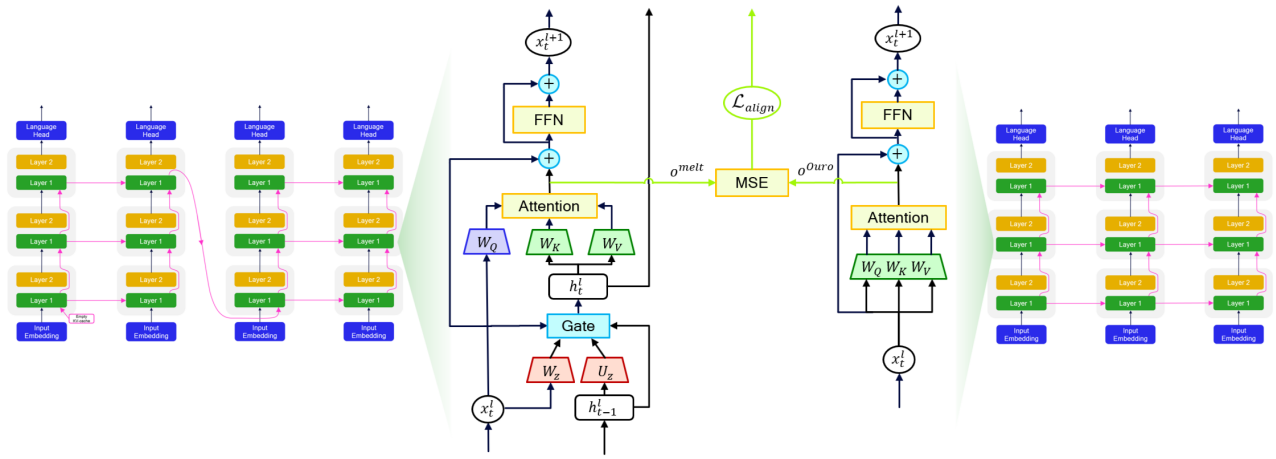


Figure 5. The **auxiliary alignment loss** matches MELT attention outputs to the corresponding outputs of the frozen LoopLM teacher at each layer and reasoning loop..

D. Hyperparameters

This section provides the hyperparameters required to reproduce our training and evaluation runs. Table 6 reports the hyperparameters used for MELT-1.6B and the ablation studies, while Table 7 summarizes the benchmark evaluation settings. Note that for HLE, we evaluate on a 300-sample subset of the original dataset.

Table 6. Training hyperparameters for **MELT-1.6B**.

Parameter	**Value**
Dataset-mix	50% AceReason-1.1-SFT, 50% OpenThoughts3
# layers	24
Hidden dimension	2048
Recurrent steps	4
# gating params.	$24 \times 2048^2 \times 2 \approx 0.2$B
Original params. init.	Ouro-1.4B-Thinking
Chunk size (tokens)	500
Batch size (tokens)	320K
Seq. length (tokens)	10K
Gradient norm clipping	1.0
LR scheduler	Cosine decay with warmup
Optimizer	Adam (β_1=0.9, β_2=0.95)
Weight decay	1.0×10^{-4}
Learning rate	8×10^{-6}
Gate learning rate	5.0×10^{-4}
Phase 1	
Interpolation training steps	500
Training tokens	160M
Phase 2	
Training steps	300
Training tokens	96M
Attention Aligned Loss β	0.1

Regarding compute, the main training run for MELT-1.6B required 130 hours on a node with 8 H100 GPUs (80GB), corresponding to 1,040 GPU-hours. The three ablation runs each used only the first training phase and required 60 hours on

Table 7. Benchmarks information

Benchmark	Number of samples	Reported metrics	Number of completions
AIME24	30	pass@1, pass@10	16
AIME25	30	pass@1, pass@10	16
AIME26	30	pass@1, pass@10	16
AMC23	40	pass@1, pass@10	16
MATH-500	500	accuracy	1
OlympiadBench	581	accuracy	1
GPQA	448	accuracy	1
HLE	300	accuracy	1
MMLU-Red	5700	accuracy	1
Humaneval	164	accuracy	1

the same 8-GPU node, for a total of 1,440 GPU-hours. Evaluation required approximately 500 GPU-hours, while preliminary experiments accounted for roughly 15,000 GPU-hours. Overall, the project used approximately 20,000 GPU-hours.

E. Theoretical analysis

As described in Section 3, MELT updates only the last row of the state matrix. Therefore, we specialize our proofs to take that aspect explicitly into account. Notwithstanding, our results are fully generalizable to the case of the full matrix update.

E.1. Spectral stability of the gated update

The next proposition shows that, in the saturated-gate regime, gradients are preserved across loops, providing the foundation for the Gradient Superhighway described in Subsection 3.2.

Proposition E.1 (Spectral Regulation in the Saturated Regime)**.** *Let $H_t^{(i)} \in \mathbb{R}^{L\times D}$ be the latent KV state at iteration t for layer i, where L is the sequence length and D is the feature size. Let us consider h_t as the $D \times 1$ state vector corresponding to the current token in the sequence (we omit the layer index for ease of notation). Consider MELT's element-wise update rule:*

$$h_t = z_t \odot h_{t-1} + (\mathbf{1} - z_t) \odot x_t \tag{1}$$

where $x_t \in \mathbb{R}^{D\times 1}$ is the hidden state at the current layer given input token x, and $z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \in \mathbb{R}^{D\times 1}$ is the gate vector. If the gating mechanism saturates such that $z_t \to \mathbf{1}$ (element-wise) for a set of latent dimensions, then the Jacobian $J_t = \frac{\partial h_t}{\partial h_{t-1}}$, restricted to these dimensions, converges to the identity matrix. Consequently, the spectral radius $\rho(J_t) \to 1$. Thus, ensuring that the gradient magnitude is preserved over any arbitrary number of iterations.

Proof. The Jacobian of the update rule with respect to h_{t-1} is derived via the product rule applied to the Hadamard product in Equation 1:

$$J_t = \underbrace{\text{diag}(z_t)}_{\text{Term 1}} + \underbrace{\text{diag}(h_{t-1} - x_t)\frac{\partial z_t}{\partial h_{t-1}}}_{\text{Term 2}} + \underbrace{\text{diag}(\mathbf{1} - z_t)\frac{\partial x_t}{\partial h_{t-1}}}_{\text{Term 3}} \tag{2}$$

We analyze the limit behavior in the shielding regime where $z_t \to \mathbf{1}$:

1. **Term 1:** Approaches the identity matrix: $\lim_{z\to\mathbf{1}} \text{diag}(z_t) = \mathbf{I}$.
2. **Term 2:** The derivative of the sigmoid function $\sigma'(u) = \sigma(u)(1 - \sigma(u))$ vanishes as $z_t \to \mathbf{1}$. Thus, $\frac{\partial z_t}{\partial h_{t-1}} \to \mathbf{0}$.
3. **Term 3:** The term $(\mathbf{1} - z_t)$ approaches $\mathbf{0}$, nullifying the contribution of the recurrent weight matrix in $\frac{\partial \tilde{h}_t}{\partial h_{t-1}}$.

Consequently: $\lim_{z\to 1} J_t = \mathbf{I} + \mathbf{0} + \mathbf{0} \implies J_t \approx \mathbf{I}$. Since the eigenvalues of the identity matrix are all 1, the spectral radius is $\rho(J_t) = 1$. □

Proposition E.1 gives more insights into the role of the gate z_t. Rather than simply selecting information, it acts as a structural stabilizer for the learning process. By explicitly controlling the decay rate of the hidden state, z_t maintains the spectral radius of the recurrence dynamics near unity. This allows gradients to propagate through long sequences without vanishing, while the strict boundedness of the gate prevents the instability associated with exploding gradients.

Remark E.2 (Relaxation to Continuous Regime). In practice, the gate z_t is modeled via a sigmoid function and takes values in $[0, 1]$. While it cannot reach exactly $\mathbf{1}$, the network can learn weights such that $z_t \geq \mathbf{1} - \epsilon$ for an arbitrarily small $\epsilon > 0$. In this regime, the gradient magnitude at step T scales as $(1 - \epsilon)^T$. Provided that $\epsilon \ll 1/T$, the signal degradation is negligible over the task's reasoning horizon. Thus, the continuous gate achieves *Effective Spectral Regulation*, approximating the ideal stability derived in Proposition E.1.

Our architecture establishes a retrieval hierarchy that decouples positional addressing from feature extraction. Because the state matrix H maintains the history of processed tokens as discrete columns $\{h_1, \ldots, h_L\}$, the model effectively possesses random access to the sequence axis. The attention mechanism first employs the Value projection W_V to address the Disentanglement problem: since each token vector encodes multiple attributes in linear superposition, W_V acts as a spectral filter, to isolate the specific feature subspace required for the current computation, ensuring that only the relevant signal is propagated while orthogonal interference is suppressed.

E.2. Gradient Superhighway

Stable gradient flow is essential for effectively optimizing the early loop iterations, so our architecture is designed to avoid both vanishing and exploding gradients. To analyze this behavior, we examine how the loss $\mathcal{L}$ backpropagates to the initial state $h_0^{(l)}$ across the T recurrent updates. By the chain rule, the gradient decomposes into a product of Jacobians:

$$\frac{\partial \mathcal{L}}{\partial h_0^{(l)}} = \frac{\partial \mathcal{L}}{\partial h_T^{(l)}} \prod_{t=1}^{T} \frac{\partial h_t^{(l)}}{\partial h_{t-1}^{(l)}} = \frac{\partial \mathcal{L}}{\partial h_T^{(l)}} \prod_{t=1}^{T} J_t$$

Leveraging Proposition E.1, we observe that for latent dimensions in the shielding regime ($Z_t \approx \mathbf{1}$), the local Jacobian effectively acts as the identity operator ($J_t \approx \mathbf{1}$). This simplifies the product significantly:

$$\prod_{t=1}^{T} J_t \approx \prod_{t=1}^{T} \mathbf{1} = \mathbf{1}$$

Such a behavior establishes a *Gradient Superhighway*: a direct path that allows error signals to traverse arbitrary depths. In contrast to standard recurrent dynamics, where gradient norms typically scale as $\mathcal{O}(\lambda^T)$—inevitably leading to exponential decay for spectral radii $|\lambda| < 1$—our architecture ensures that $\|\frac{\partial \mathcal{L}}{\partial h_0^{(l)}}\| \approx \|\frac{\partial \mathcal{L}}{\partial h_T^{(l)}}\|$. This structural stability effectively alleviates the vanishing gradient problem, enabling the optimization of deeper looped transformer models.

F. Notes on reproducibility and inference efficiency in Ouro

This appendix documents several technical observations we made while attempting to reproduce and analyze claims from (Zhu et al., 2025). Our goal is not to diminish the contributions of Ouro, but to clarify practical aspects that are directly relevant when comparing memory usage and inference efficiency against MELT.

F.1. Reproducibility

Despite substantial effort, we were unable to fully reproduce the reported results of Ouro under the configurations described in the paper. While the authors provide the model's code and pretrained checkpoints, key implementation details and evaluation settings are either underspecified or differ from what is required to match the reported numbers. As a result, we observed non-trivial discrepancies between the performance reported in the paper and the results obtained using the released artifacts. Therefore, throughout this work we rely exclusively on values obtained from our own experimental measurements. Notably, even under these measurements, Ouro remains competitive and continues to outperform the state of the art, underscoring the strength of its underlying approach.

F.2. Early-exit gating and effective compute

A central component of Ouro is its learned gating mechanism, which is intended to enable adaptive computation by allowing tokens to exit early when additional recurrent steps are deemed unnecessary. While this mechanism is emphasized throughout the paper, we observed the following in practice:

- Ouro introduces an early-exit mechanism, but the released default configuration uses a threshold that effectively disables early exiting, and the paper does not specify how this threshold should be chosen in practice.
- Even when an early exit is triggered, the model still executes all recurrent loop computations up to the maximum depth; the gating affects which logits are selected, not whether subsequent loops are computed.

We verified this behavior by inspecting the released inference code. As a consequence, the gating mechanism does not reduce inference-time compute or memory usage under typical settings, despite its conceptual framing as an adaptive compute mechanism.

We hypothesize that this design choice is driven by KV-cache dependencies: since later tokens require access to the KV states produced at the final loop, it is not possible to terminate computation early for a given token without breaking autoregressive consistency. It is worth noting that this limitation would not apply to MELT's constant-memory KV update mechanism, although we leave a full investigation of early-exit strategies in MELT to future work.

G. Existing assets

In this appendix, we provide a comprehensive overview of all assets used throughout this work, along with their corresponding licenses to ensure transparency and reproducibility. Specifically, we list the evaluation benchmarks (Table 8), the models considered in our comparisons (Table 9), the training datasets employed (Table 10), and the main codebases used in our implementation (Table 11). All resources are referenced with links to their original sources.

Table 8. The list of benchmarks.

Benchmarks	Link	License
MATH500 (Lightman et al., 2023)	HuggingFace	MIT
AIME 2024 (Mathematical Association of America, 2024)	HuggingFace	Copyright MAA
AIME 2025 (Mathematical Association of America, 2025)	HuggingFace	Copyright MAA
AIME 2026 (Mathematical Association of America, 2026)	HuggingFace	Apache 2.0
OlympiadBench (He et al., 2024)	HuggingFace	MIT
AMC 2023 (Mathematical Association of America, 2023)	HuggingFace	Copyright MAA
GPQA (Rein et al., 2023)	HuggingFace	CC BY 4.0
HLE (Phan et al., 2026)	HuggingFace	MIT
MMLU (Hendrycks et al., 2021a)	HuggingFace	MIT
HumanEval (Chen et al., 2021c)	HuggingFace	MIT

Table 9. The list of models.

Model	Link	License
Ouro-Thinking-1.4B (Zhu et al., 2025)	HuggingFace	Apache 2.0
Ouro-Thinking-2.6B (Zhu et al., 2025)	HuggingFace	Apache 2.0
Qwen3-1.7B (Yang et al., 2025)	HuggingFace	Apache 2.0
Gemma4-E2B (Google, 2024)	HuggingFace	Apache 2.0
Qwen3.5-2B (Team, 2026)	HuggingFace	Apache 2.0
DeepSeek-R1-1.5B (Guo et al., 2025)	HuggingFace	MIT

Table 10. The list of training datasets.

Dataset	Link	License
AceReason-1.1-SFT (Liu et al., 2025)	HuggingFace	cc-by-4.0
OpenThoughts3 (Guha et al., 2025)	HuggingFace	Apache 2.0

Table 11. The main codebases used.

Codebase	Link	License
Transformers (Wolf et al., 2020)	GitHub	Apache 2.0
TRL (von Werra et al., 2020)	GitHub	Apache 2.0
vLLM (Kwon et al., 2023)	GitHub	Apache 2.0
LightEval (Habib et al., 2023)	GitHub	MIT