# HRM-Text: Efficient Pretraining Beyond Scaling

Guan Wang[1,*,†], Changling Liu[1,*], Chenyu Wang[2], Cai Zhou[2], Yuhao Sun[1],
Yifei Wu[1], Shuai Zhen[1], Luca Scimeca[1], Yasin Abbasi Yadkori[1,†]

[1]Sapient Intelligence [2]MIT

**Abstract**

The current pretraining paradigm for large language models relies on massive compute and internet-scale raw text, creating a significant barrier to foundational research. In contrast, biological systems demonstrate highly sample-efficient learning through multi-timescale processing, such as the functional organization of the frontoparietal loop. Taking this as inspiration, we introduce HRM-Text, which replaces standard Transformers with a Hierarchical Recurrent Model (HRM) that decouples computation into slow-evolving strategic and fast-evolving execution layers. To stabilize this deep recurrence for language modeling, we introduce *MagicNorm* and warmup deep credit assignment. Furthermore, instead of standard raw-text pretraining, we train exclusively on instruction-response pairs using a task-completion objective and PrefixLM masking. Serving as an empirical existence proof of efficient pretraining, a 1B-parameter HRM-Text model trained from scratch on only 40 billion unique tokens and $1,500 budget achieves 60.7% on MMLU, 81.9% on ARC-C, 82.2% on DROP, 84.5% on GSM8K, and 56.2% on MATH. Despite utilizing roughly 100-900x fewer training tokens and 96-432x less estimated compute than standard baselines, HRM-Text performs competitively with 2–7B parameter open models. These results demonstrate that co-designing architectures and objectives can radically reduce the compute-to-performance ratio, making pretraining from scratch accessible to the broader research community.

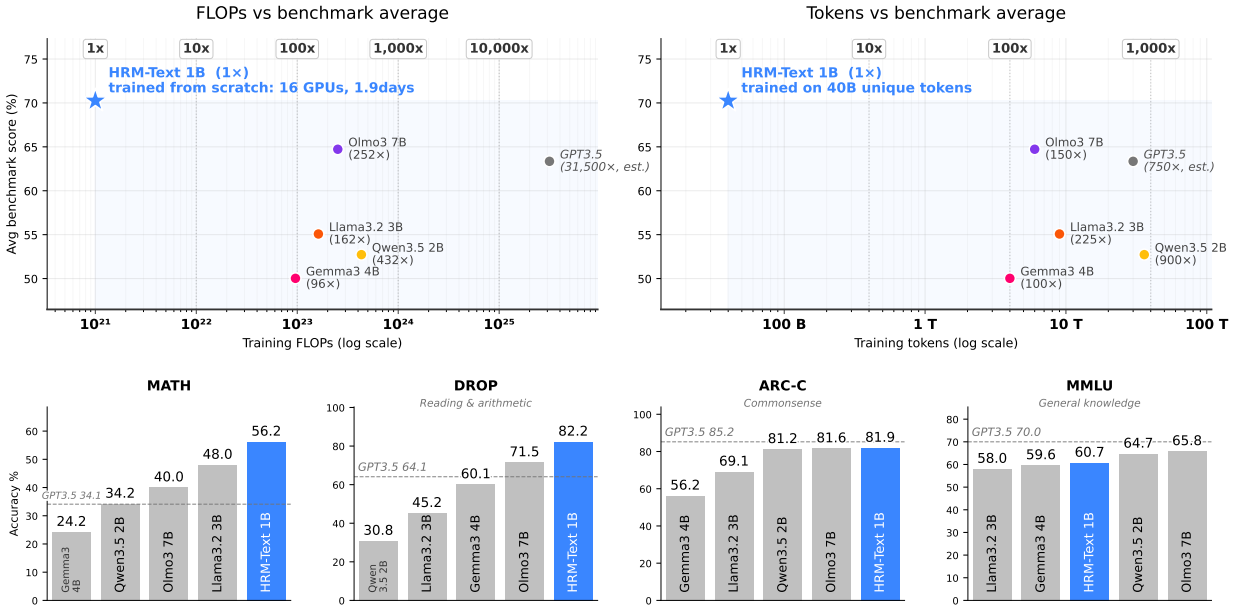



Figure 1: **Pretraining efficiency.** Trained from scratch in 1.9 days on 16 GPUs, HRM-Text 1B achieved performance competitive with substantially larger 2–7B foundation models while utilizing up to 432× less compute and 900× fewer training tokens.

† Corresponding author. * Equal Contribution. Contact: research@sapient.inc.
Code available at: github.com/sapientinc/HRM-Text

# 1 Introduction

The remarkable success of large language models (LLMs) is currently driven by a monolithic recipe: massive, multi-stage pipelines that begin with broad unsupervised pretraining over internet-scale raw text. While undeniably effective, this brute-force scaling paradigm is highly inefficient in data-limited regimes. Massive compute is spent predicting prompt-like or task-irrelevant text simply to build generalized representations[1,2,3]. Consequently, this extreme computational barrier has largely locked the broader research community out of foundational pretraining exploration. The prevailing assumption is that without immense compute clusters and trillions of tokens, investigating new architectures or training from scratch is futile.

This brute-force data hunger stands in stark contrast to human intelligence, which can grasp governing rules and perform heuristic-guided search from only a few examples. In our previous work, we introduced the Hierarchical Recurrent Model (HRM), a dual-timescale architecture inspired by the functional organization of the biological frontoparietal loop[4]. By decoupling deliberation into a slow-evolving strategic layer and a fast-evolving execution layer, HRM provided a structural inductive bias that helped avoid local stagnation and successfully guided symbolic search on combinatorial tasks.

However, scaling recurrent architectures to the open-ended complexities of language modeling introduces severe gradient-instability risks[5,6,7,8]. A structural prior alone is insufficient; achieving competitive open-domain performance requires a holistic codesign. In this paper, we demonstrate that architecture and training methods are profoundly important once again. We explore two major, synergistic directions to realize this sample-efficient engine:

- **Architectural Exploration:** To achieve deep computation without a proportional explosion in parameter counts, we build upon HRM's modular, multi-timescale recurrence. The fast $L$-module performs local iterative refinement, while the slow $H$-module maintains stable semantic context across cycles[4]. To make this deep recurrence mathematically viable for language, we introduce stabilization techniques like *MagicNorm* and warmup deep credit assignment, which bound forward activation variance while maintaining backward optimization stability[9,10,11].
- **Objective Exploration:** We challenge the dogma of autoregressive pretraining on raw text. Since models are primarily used for conditional generation at inference time, we pretrain HRM-Text directly from scratch on instruction-response pairs[12,13,14]. We optimize a task-completion objective, computing the negative log-likelihood loss exclusively over the response: $-\log P(x_a \mid x_q)$[15,16,13]. We pair this with a PrefixLM attention mask, which allows full bidirectional (encoder-like) attention across the instruction tokens while preserving standard causal generation for the response[17,18,16,3].

When these two directions are combined, the result is an empirical existence proof that defies the current scaling dogma. Trained from scratch on a low budget of only 40B unique tokens, HRM-Text achieves strong performance on most benchmarks against contemporary open models like Llama, Qwen, Gemma, OLMo, Ouro and Huginn[19,20,21,22,23,24]. Strikingly, it reaches this performance neighborhood using roughly $100\text{-}900\times$ fewer training tokens and $96\text{-}432\times$ less estimated training compute than these baselines, as shown in Figure 1 and Table 4.

We do not present HRM-Text as the final or optimal language model, but rather as proof that specific structural priors and targeted training objectives can radically alter the compute-to-performance

ratio. Because the entry price is vastly reduced, this methodology democratizes foundational AI research. Pretraining from scratch is accessible again—we invite the community to join us in exploring how far smart architectures and focused objectives can go.

# 2 Methods

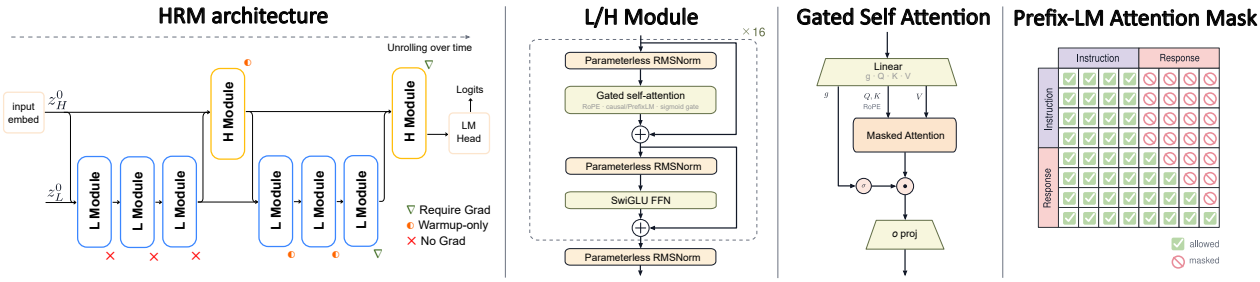



Figure 2: **HRM-Text architecture.** (a) Dual-timescale recurrent design comprising L and H modules. (b) L/H module internals featuring *MagicNorm*—PreNorm blocks followed by final norm. (c) Sigmoid-gated multi head self-attention. (d) PrefixLM mask enabling bidirectional attention on instruction.

HRM-Text builds upon an improved HRM architecture, featuring a dual-timescale recurrence[4]. The forward pass is initialized with a high-level state, $z_H^0$, derived from the input token embeddings, alongside a fixed low-level state, $z_L^0$. The core processing sequence consists of two high-level cycles. Each cycle executes three fast $L$ module updates followed by a single slow $H$ module update. Token logits are generated by applying a linear head to the output of the final $H$ module state. We employ a warmup deep credit assignment strategy: gradients are initially backpropagated through only the final two recurrent steps, expanding to the final five steps as training progresses.

Internally, both the $H$ and $L$ recurrent modules are structured using *MagicNorm*. Additionally, we utilize parameterless RMSNorm (omitting the learnable $\gamma$ parameter)[25], SwiGLU activation functions[26], Rotary Position Embeddings (RoPE)[27], and a sigmoid-gated self-attention mechanism[28].

In contrast to standard autoregressive pretraining on raw text, we optimize a task-completion objective. The model is pretrained directly on instruction-response pairs $(x_q, x_a)$ from scratch using a negative log-likelihood (NLL) loss computed exclusively over the response, $-\log P(x_a|x_q)$. This objective is naturally paired with a PrefixLM attention mask, enabling full bidirectional attention across the instruction tokens.

In the following sections, we detail the specific mechanics that enable HRM-Text's extreme efficiency. Section 2.1 delves into our novel stabilization techniques, while Section 2.2 explores the task-completion pretraining objective and PrefixLM masking strategy.

## 2.1 Scaling to language with recurrence

### 2.1.1 Stabilization via *MagicNorm*

Although the original HRM demonstrated strong performance on symbolic tasks, scaling recurrent architectures to language modeling introduces severe gradient-instability risks. Transformer design

already involves a compromise in the placement of normalization layers[9,10]; recurrence amplifies this compromise because the same transformation is repeatedly applied over many steps.

**PostNorm**[29] places the normalization outside the residual branch:

$$h_l = \text{Norm}(h_{l-1} + \text{Sublayer}(h_{l-1}))$$

This effectively bounds activation variance and can improve expressivity, but it disrupts the clean identity path and can lead to vanishing gradients in deeper networks[10].

**PreNorm** places the normalization inside the residual branch:

$$h_l = h_{l-1} + \text{Sublayer}(\text{Norm}(h_{l-1}))$$

This maintains a direct identity path, $h_L = h_0 + \sum_{l=1}^{L} \text{Sublayer}(\cdot)$, allowing gradients to flow more directly to early layers. However, the unnormalized residual accumulation can cause hidden-state variance to grow with depth, which may lead to representation collapse or reduced performance relative to PostNorm.

***MagicNorm*:** To address this tradeoff in recurrent models, we introduce *MagicNorm*, which exploits the asymmetry between the forward and backward computational horizons induced by truncated backpropagation through time (TBPTT).

Let $N$ denote the total number of recurrent forward steps and $K$ denote the truncated backward horizon, where $K \ll N$. In *MagicNorm*, each recurrent module is composed of $L$ internal PreNorm blocks, but is capped with a final normalization layer at its exit:

$$z_n = \text{Norm}\left( z_{n-1} + \sum_{l=1}^{L} \text{Sublayer}_l(\text{Norm}(\cdot)) \right)$$

During the *forward pass*, the recurrent state $z$ is subjected to $N$ module-level normalization operations. Because these norms sit directly on the main recurrent pathway, they bound activation variance at the end of every recurrent step. This prevents the unbounded variance growth of pure PreNorm and gives the recurrent core PostNorm-like forward stability.

Conversely, during the *backward pass*, the truncated gradient horizon means the error signal passes through the module-level normalization only $K$ times. Within that same horizon, the gradient also flows through $L$ internal PreNorm identity connections. Since $K$ is small relative to the full recurrence depth $N$, MagicNorm behaves more like a stable PreNorm architecture during optimization.

### 2.1.2 Warmup deep credit assignment

The original HRM uses a fixed 1-step gradient strategy, backpropagating only through the last two recurrent steps (last $H$ and last $L$). We extend this approach with *warmup deep credit assignment*. The schedule is motivated by temporal-curriculum principles: early optimization is restricted to short credit-assignment paths, and longer paths are introduced only after the model has reached a more stable regime. This design is also consistent with biological accounts of temporal learning, where local traces can support delayed credit assignment[30], reward-predictive signals can shift from reward-proximal events to earlier cues[31], and developmental curricula can improve sequence learning by exposing learners to shorter-range structure before longer-range dependencies[32].

Operationally, we dynamically adjust the backward gradient horizon, $K$. During early pretraining, we compute gradients through only the last two recurrent steps ($K = 2$), then linearly warm up

the horizon to the last five steps ($K = 5$). This progressive deepening allows the model to exploit longer recurrent computation while reducing exposure to the optimization pathologies that often arise from long gradient paths at initialization. Because the warmup phase backpropagates through fewer recurrent steps than the final setting, it also reduces the average backward-pass computation and accelerates early training.

## 2.2 Task-completion objective and PrefixLM

The dominant paradigm for training foundation models relies on a resource-intensive, multi-stage pipeline. From T5 through modern large language models[16], training typically begins with broad unsupervised pretraining and is followed by higher-quality mid-training.

In the pretraining phase, models are trained on internet-scale raw corpora to learn general language representations. In the mid-training (or annealing) phase, the model is refined on high-quality text, usually instruction-like data. In both phases, the model optimizes an NLL objective over all tokens

$$-\log P(x)$$

While effective, this approach can be inefficient in the data- and resource-limited regime. Broad raw-text pretraining consumes most of the compute and data, and much of the token-level loss is spent on predicting prompt-like or task-irrelevant text. Yet at inference time, models are applied primarily on conditional generation: *given a query or instruction, they must produce an appropriate response.*

To improve sample efficiency, HRM-Text omits broad raw-text pretraining and trains exclusively on instruction-response pairs from scratch. Given an example containing an instruction and response $x = (x_q, x_a)$, we optimize the NLL of the response conditioned on the instruction:

$$-\log P(x_a|x_q)$$

By not predicting the instruction tokens, the model concentrates its parameter updates on generating accurate responses. Figure 3-(a) illustrates this effect. Although the total loss is comparable with and without the task-completion objective, the error associated with the response component is substantially lower.

Furthermore, this single-stage conditional objective naturally aligns with a PrefixLM attention mask[16]. Because the model is never required to autoregressively predict the instruction $x_q$, we remove the causal masking over the instruction segment: all instruction tokens attend to one another bidirectionally, while standard causal masking is maintained over the response sequence. This gives HRM-Text an encoder–decoder-like separation inside a decoder-style implementation. The instruction segment is first integrated as a fully visible context, analogous to an encoder-side representation, while the response segment is generated autoregressively, analogous to a decoder.

Figure 3 (b) shows that PrefixLM leads to higher attention softmax entropy, indicating attention over a more diverse set of tokens. Figure 3 (c) shows that causal attention is more localized, whereas PrefixLM attention is more global and diverse. Together, the response-only conditional loss and PrefixLM attention improve sample efficiency in the data- and compute-restricted regime.

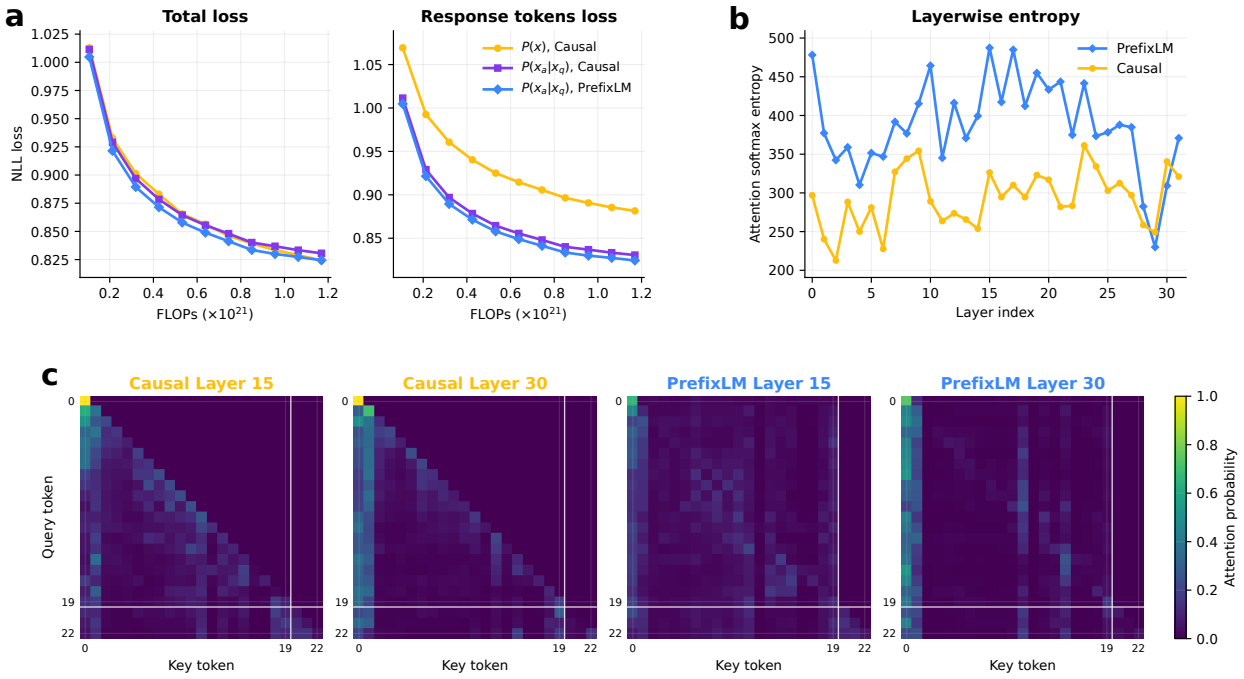



Figure 3: **Task-completion and PrefixLM improve response modeling.** (a) Compared with full causal language modeling $P(x)$, response-only training $P(x_a|x_q)$ lowers response-token NLL. PrefixLM further improves response loss. (b) PrefixLM increases layerwise attention entropy relative to causal masking, suggesting broader use of the prompt. (c) Attention maps illustrate the qualitative difference: causal attention remains mostly local and triangular, while PrefixLM enables global bidirectional interactions among prompt.

# 3 Results

As the central question of this paper is whether a model trained from random initialization under a small pretraining budget can reach a meaningful open-model performance regime, we approach this question as a small-budget design exploration: first, whether architectural choices can improve the use of fixed training compute, and second, whether the objective and input structure can increase the yield of each training example. Finally, we compare HRM-Text with contemporary fully open and open-weight models to quantify its efficiency relative to current pretraining practice, and analyze whether the recurrent architecture increases effective depth. Training details for all models are provided in Section 4.

Across these experiments, HRM-Text is trained from scratch on the task-formatted mixture described in Section 4.1, using only 40B unique tokens. We report all the performance from a single HRM-Text checkpoint.

## 3.1 Architecture efficiency under matched training compute

The first part of this exploration asks how much architecture design can improve the use of a fixed training budget. We test this by comparing standard Transformers, larger matched-FLOPs Transformers, Looped Transformers[33], RINS[34], and HRM under matched training compute.

| Model | Recursions | FLOPs ($10^{21}$) | Tokens (T) | MMLU | ARC-C | Hella. | Wino. | BoolQ | DROP | GSM8K | MATH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HRM 1B | 4 (H2L3) | 1.0 | 0.06 | **60.73** | **81.91** | **63.42** | **72.38** | **86.18** | **82.21** | **84.53** | **56.16** |
| Looped Transformer 1B | 4 | 1.0 | 0.06 | 56.51 | 74.06 | 47.59 | 68.19 | 82.35 | 76.20 | 75.13 | 48.30 |
| RINS 1B | 4 ($r = 7$) | 1.1 | 0.06 | 56.09 | 76.71 | <u>52.49</u> | <u>68.35</u> | 83.98 | <u>79.92</u> | <u>77.71</u> | 48.90 |
| Transformer 1B | 1 | 1.0 | 0.17 | 53.15 | 74.32 | 47.29 | 65.51 | 83.58 | 75.30 | 75.06 | 48.36 |
| Transformer 3B (Deep) | 1 | 1.1 | 0.06 | <u>56.67</u> | <u>80.46</u> | 51.05 | 67.96 | <u>84.07</u> | 76.95 | 75.66 | <u>50.50</u> |
| Transformer 3B (Wide) | 1 | 1.1 | 0.06 | 54.53 | 76.54 | 43.77 | 67.09 | 83.55 | 74.04 | 73.01 | 49.74 |

Table 1: Training FLOPs-matched comparison of recurrent architectures and Transformer models. Bold denotes the highest score in each column, and underline denotes the second highest.

Table 1 compares training-FLOPs-matched recurrent architectures (including HRM, looped Transformers, and RINS) with standard Transformers. For recursive models, the value in the recursions column indicates total compute per forward pass, expressed as a multiple of the compute required if recurrence is not present. For example, H2L3 denotes 2 outer H cycles, with 3 L steps inside each outer cycle, giving $2 \times (3 + 1) = 8$ total H/L module steps. Since each H or L module contains half of the non-embedding parameters of the full HRM recurrent core, this corresponds to $8 \times 0.5 = 4$ recursions in the table. For standard Transformer models, the value is 1.

Looped Transformers and RINS generally outperform Transformer models of the same size, showing that recurrent or looped computation is an effective architectural direction. When compared with a larger Transformer under a matched training-FLOPs budget, however, their advantage is less consistent. HRM is a strong instance of this architecture-design space and performs well against the listed baselines, including the larger deep Transformer.

Within recurrent designs, we further compare HRM with TRM to separate hierarchical dual-timescale recurrence from a shared-parameter dual-timescale recurrent variant.

| Model | Recursions | FLOPs ($10^{21}$) | Tokens (T) | MMLU | ARC-C | Hella. | Wino. | BoolQ | DROP | GSM8K | MATH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HRM 1B | 4 (H2L3) | 1.0 | 0.06 | **60.73** | **81.91** | **63.42** | **72.38** | **86.18** | **82.21** | **84.53** | **56.16** |
| TRM 1B (*unstable*) | 2 (H2L1) | 1.1 | 0.06 | 46.38 | 56.66 | 35.22 | 59.35 | 74.74 | 63.56 | 67.63 | 44.12 |
| HRM 0.6B | 4 (H2L3) | **0.5** | 0.06 | 56.87 | 74.32 | 50.48 | 66.38 | <u>84.86</u> | 78.07 | 78.39 | 50.28 |
| TRM 0.6B | 4 (H2L3) | 1.1 | 0.06 | <u>56.88</u> | <u>76.71</u> | <u>55.12</u> | <u>67.01</u> | 84.22 | <u>79.91</u> | <u>78.47</u> | <u>50.84</u> |

Table 2: **Performance and stability comparison against TRM.** HRM maintains stable training dynamics across all scales, whereas TRM suffers from severe instability at the 1B parameter scale. Furthermore, at the 0.6B scale, HRM achieves competitive performance across most benchmarks while requiring 2× less compute than TRM.

TRM is a HRM-variant that shares the H and L module parameters, to achieve strong results on symbolic reasoning problems at smaller scale[35]. Table 2 compares HRM and TRM. Since TRM shares parameters across H-L modules, there are two ways to approximately match FLOPs: keeping the overall parameter count fixed and reduce the number of recursions, or keeping the recursive structure fixed and reduce the parameter count. In the first setting, TRM training is less stable, likely due to the reduced recursion weakening the intended iterative computation. In the second

setting, the additional recursion stabilizes training and improves performance, but the model still lags behind FLOPs-matched HRM. HRM achieves generally comparable or stronger performance while using substantially fewer FLOPs than TRM in this comparison.

These results support the first part of the small-budget design exploration: recurrent and looped architectures can improve benchmark yield under fixed training compute, and HRM is one effective point in this broader architecture-design space.

## 3.2 Task-completion objective and PrefixLM yield

The second part of this exploration asks whether the training objective and input structure can increase the yield of each training example. We test this through an incremental ablation that starts with a standard Transformer trained on full question–answer pairs using causal attention, then adds the task-completion objective, PrefixLM attention, and finally the HRM architecture. All experiments are FLOPs-matched.

As shown in Table 3, the task-completion objective, PrefixLM training, and the HRM architecture each significantly contribute to overall performance. Introducing the task-completion objective establishes initial gains across all benchmarks, while PrefixLM training further enhances these results compared to standard causal masking. Ultimately, transitioning from a standard Transformer to the HRM architecture delivers a final, consistent performance increase across the board.

| Architecture | Objective | Attention | MMLU | ARC-C | Hella. | Wino. | BoolQ | DROP | GSM8K | MATH |
|---|---|---|---|---|---|---|---|---|---|---|
| Transformer 1B | $P(x)$ | Causal | 40.55 | 51.91 | 32.50 | 48.54 | 39.94 | 38.24 | 48.37 | 35.44 |
| | $P(x_a\|x_q)$ | Causal | 47.72 | 62.88 | 41.64 | 60.85 | 76.57 | 54.24 | 69.75 | 47.04 |
| | $P(x_a\|x_q)$ | PrefixLM | 53.15 | 74.32 | 47.29 | 65.51 | 83.58 | 75.30 | 75.06 | 48.36 |
| HRM 1B | $P(x)$ | Causal | 43.68 | 60.24 | 45.10 | 53.71 | 55.98 | 42.74 | 66.19 | 44.32 |
| | $P(x_a\|x_q)$ | Causal | 50.60 | 69.80 | 50.43 | 63.93 | 67.28 | 62.39 | 79.91 | 54.18 |
| | $P(x_a\|x_q)$ | PrefixLM | **60.73** | **81.91** | **63.42** | **72.38** | **86.18** | **82.21** | **84.53** | **56.16** |

Table 3: Performance Comparison across Model Architectures and Objectives

## 3.3 Comparison with contemporary open models

After exploring architecture, objective, and input structure under the small-budget setting, we compare the resulting HRM-Text checkpoint with contemporary fully open and open-weight models trained with substantially larger budgets.

Figure 1 and Table 4 compares HRM-Text 1B with contemporary fully open and open-weight models, including Llama, Qwen, Gemma, OLMo and recurrent models, Huginn and Ouro. HRM-Text achieves strong performance among these models on most benchmarks, while remaining competitive on MMLU despite its smaller parameter count and limited 40B unique-token pretraining budget. This pattern is consistent with the role of HRM-Text: recurrent depth and task-completion pretraining improve reasoning and task execution, while broad factual-knowledge coverage remains more sensitive to model scale and data breadth. HRM-Text reaches this performance range with 96-432$\times$ less estimated training compute and roughly 100-900$\times$ fewer training tokens than the compared open baselines. This comparison supports the paper's central question by showing that

| Model | Architecture | FLOPs($10^{21}$) | Tokens(T) | MMLU | ARC-C | Hella. | Wino. | BoolQ | DROP | GSM8K | MATH |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Fully open* | | | | | | | | | | | |
| HRM-Text 1B | Recurrent | 1 | 0.06 | 60.7 | **81.9** | 63.4 | **72.4** | **86.2** | **82.2** | **84.5** | **56.2** |
| Huginn 3.5B | Recurrent | 127 | 0.8 | 31.4 | 38.2 | 65.2 | 59.4 | 69.8 | 17.8 | 34.6 | 12.6 |
| Olmo3 7B | Dense | 252 | 6 | 65.8 | 81.6 | 72.7 | 64.6 | 85.4 | 71.5 | 75.5 | 40.0 |
| *Open weight* | | | | | | | | | | | |
| Llama3.2 3B | Dense | 162 | 9 | 58.0 | 69.1 | 47.1 | 52.4 | 76.2 | 45.2 | 77.7 | 48.0 |
| Gemma3 4B | Dense | 96 | 4 | 59.6 | 56.2 | **77.2** | 64.7 | 72.3 | 60.1 | 38.4 | 24.2 |
| Qwen3.5 2B | Dense | 432 | 36 | 64.5 | 81.0 | 64.6 | 56.7 | 80.5 | 30.8 | 53.0 | 34.2 |
| Ouro 1.4B | Recurrent | 259 | 7 | **67.4** | 60.9 | 74.3 | 72.3 | 83.6 | 49.7 | 78.9 | 22.4 |

Table 4: Evaluation results of HRM-Text 1B and contemporary fully open or open-weight models.

a small, task-completion-oriented pretraining run can enter the performance range of open models trained with far larger token and compute budgets.

Our reported scaling experiments extend to 3B parameters for Transformers and 1B parameters for HRM-Text. Within this range, the results show that models trained with a limited amount of data can remain competitive with contemporary industrial-scale pretraining efforts that use much larger datasets (up to 36T tokens). Demonstrating similar efficiency gains at larger model scales remains in the scope of future work.

### 3.4 Effective depth analysis

We hypothesize that HRM's effectiveness is due to its recurrence, increasing the amount of useful internal computation. We test this hypothesis by examining whether HRM exhibits greater effective depth than standard and looped Transformer baselines.

Figure 4 illustrates effective depth from two perspectives: (a) the norm of the difference between adjacent recurrent blocks, and (b) the cosine similarity of block-wise representations. Both metrics suggest that HRM maintains more active representational change across depth than standard Transformers and other looped models.

Following Hu et al.[36], we also use logit lens analysis to evaluate how early the model's output distribution begins to stabilize. We decode hidden states from different layers using the model's output projection head, then compute the KL divergence between each probed prediction and the final model distribution. As shown in Figure 5, both the standard Transformer and looped Transformer converge to a stable output distribution in relatively early layers, suggesting that their deeper layers make smaller incremental contributions. In contrast, HRM retains larger KL values in deeper layers, indicating greater effective depth.

## 4 Training details

### 4.1 Dataset

We train HRM-Text exclusively on open-source datasets, comprising general instructions, rewritten knowledge, mathematical and symbolic tasks, textbook exercises, and web-extracted questions. The initial corpus contains approximately 176.5B tokens across 593.7M documents. From this, we

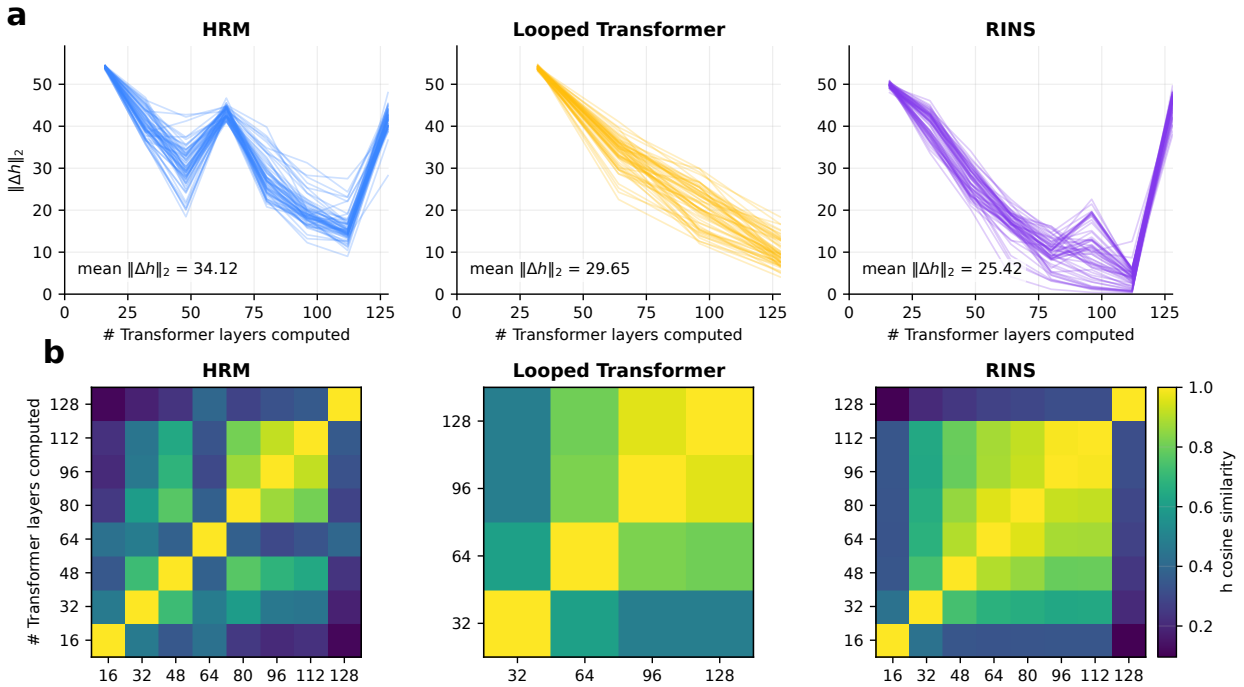



Figure 4: **Effective depth analysis**. (a) Each layer of HRM consistently reveals considerable changes compared to its previous layer, showing that deep layers of HRM are still making meaningful contributions to the hidden states. (b) HRM has smaller cosine similarity of block-wise representations, while other model variants suffer more from the common layer representation over-smoothing issue, analogously to standard transformers.

sample 40B unique tokens for a total training duration of 60B tokens, with repetition governed by the stratified sampling schedule described below. Table 5 summarizes the dataset composition.

To control response properties during inference, we prepend specific condition tags to the instructions based on the target response style. We utilize four primary conditions: `direct` (answer-only), `cot` (chain-of-thought), `synth` (synthetic answer style), and `noisy` (web-crawl text with uneven formatting). As outlined in the "Condition" column of Table 5, this approach leverages conditioned training[52,53] to enable explicit selection of the model's output format at generation time.

Table 6: Stratified sampling limits used to construct the training mixture.

| Source dataset | Sampling cap |
|---|---|
| FLAN | 5k docs/task |
| Tasksource | 10k docs/task |
| SYNTH | 10M docs |
| AceReason | 2M docs |
| OpenThoughts2 | 500k docs |
| DM Math | 100k docs/task |
| Sudoku | 1M docs |
| Small datasets ($\leq$ 50k docs) | 10$\times$ original size |
| Other datasets | Original size |

To concentrate the training signal on final task completions, we strip all text enclosed within `<think>...</think>` boundaries prior to training. This eliminates explicit long-CoT traces mostly produced by reinforcement learning with verifiable rewards (RLVR) training[54], aligning with our objective for HRM-Text to rely on its internal hierarchical computation rather than explicit reasoning steps.

We employ SeqIO-style stratified sampling[55], treating each dataset or task as an independent stratum rather than sampling uniformly from a pooled corpus. To ensure a balanced training mixture

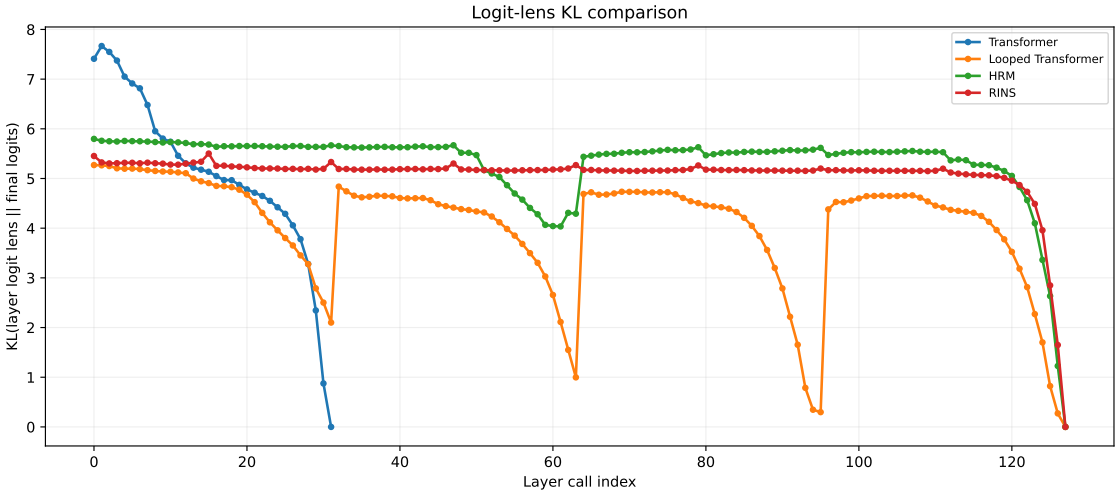



Figure 5: **Per-layer logit lens KL**. HRM shows the largest logit len KL in deep layers, while both standard and looped transformers converges to stable distributions in shallow layers.

| Type | Datasets | Tokens | Docs | Condition |
|---|---|---|---|---|
| General instructions | FLAN[14], Tasksource[37], NoRobots[38] | 138.7B | 379.9M | direct / cot |
| Rewritten Wikipedia knowledge | SYNTH[39] | 21.7B | 60.8M | synth, direct / cot |
| Math and reasoning | Platypus[40], Principia[41], OpenMathInstruct2[42], NuminaMath[43], OmniMATH[44] | 6.8B | 25.7M | synth / synth, cot |
| Symbolic | DMMath[45], AMPS[46], Sudoku-Extreme[4] | 6.2B | 120.7M | direct |
| Reasoning (thinking tokens removed) | AceReason[47], OpenThoughts2[48] | 2.4B | 3.6M | synth, cot |
| Textbook exercises | TextbookReasoning[49] | 358.1M | 1.2M | synth, cot |
| Extracted web instructions | NaturalReasoning[50], WebInstruct-verified[51], AMPS-khan[46] | 375.1M | 1.8M | noisy / direct / cot |

Table 5: Source datasets used for HRM-Text training, grouped by type.

and prevent over-representation of massive datasets, we apply caps on number of documents per task or per dataset, while upsampling smaller datasets. The specific sampling limits and multipliers are detailed in Table 6.

## 4.2 Dataset Contamination

While our pretraining data originates from widely used public sources, and many enforce decontamination measures, residual contamination may still persist considering the scale of pretraining. To rigorously assess whether our models' benchmark performance artificially benefits from exposure to test examples, we conduct a statistical test adapted from the Llama family[56].

We tokenize questions from all evaluated benchmarks (excluding few-shot exemplars) and identify $n$-gram matches against the fully tokenized pretraining corpus. A sample's contamination percentage is defined as the fraction of its tokens participating in these matched $n$-grams.

To determine if contamination inflates performance, we partition the evaluation data into four overlapping subsets based on contamination percentage: "Clean" ($< 20\%$), "Not Clean" ($\geq 20\%$), "Not

Dirty" ($< 80\%$), and "Dirty" ($\geq 80\%$). For contamination to be deemed significantly impactful, "clean" samples must perform demonstrably worse than average, while "dirty" samples perform demonstrably better. For each subset of size $k$, we compute the empirical mean performance $\bar{X}$ and the test statistic $Z_k = (\bar{X} - \mu_k)/\sigma_k$, where $\mu_k$ and $\sigma_k$ are the mean and standard deviation of the sampling distribution for size $k$. We conclude that dataset contamination provides a statistically significant performance boost only if $|Z_k| > 2$ across all four subsets.

We applied this test to HRM-Text 0.6B and 1B using $n = 13$ and $n = 20$, on all benchmarks shown in Table 4. HRM-Text 0.6B exhibited no significant contamination in either setting. HRM-Text 1B showed statistical significance on the DROP benchmark for $n = 13$ (as shown in Table 7), but not for $n = 20$. Nonetheless, HRM-Text 1B still achieves a score of $81.1$ on the strictly clean subset ($0\%$ average contamination, $5904$ samples) of DROP, indicating strong baseline generalization despite a marginal potential benefit from contamination.

Overall, these analyses show that HRM-Text's benchmark performance is unlikely artificially driven by prior exposure to test examples.

| Dataset | Subset Type | Avg. Contam. % | $n$ | $\bar{X}$ | $\mu_n$ | $Z_n$ |
|---|---|---|---|---|---|---|
| DROP ($n = 13$) | Clean | 0.0 | 5904 | 81.1 | 82.3 | $-2.53$ |
| | Not Clean | 90.5 | 3632 | 84.2 | 82.3 | 3.23 |
| | Not Dirty | 6.1 | 6576 | 80.8 | 82.3 | $-3.25$ |
| | Dirty | 97.5 | 2960 | 85.5 | 82.3 | 4.85 |

Table 7: Contamination analysis results for the DROP dataset on HRM-Text 1B.

## 4.3 Architecture and optimization details

HRM-Text 1B took 46 hours to pretrain on two 8×H100 nodes, costing around $1,472 (assuming $2 per H100 hour). We summarize the model, optimization, and infrastructure settings below.

**Tokenizer.** We employ a Byte-Pair Encoding (BPE) tokenizer with a vocabulary size of 65,536, trained using the *tokenizers* library.

**Model configuration.** Each module is a transformer comprising 16 layers, with a hidden size of 1536 and a head size of 128. We use a context size of 4,096 and RoPE positional encoding with $\theta = 10{,}000$. The RMSNorm $\epsilon$ parameter is set to $10^{-6}$. All models are trained in *bfloat16* precision, and all model weights are initialized using LeCun normal.

**Optimization.** We use the Adam-atan2 optimizer[57] with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 0.1. The learning rate is linearly warmed up over 2,000 steps and then held constant at $2.2 \times 10^{-4}$. No gradient clipping is applied. The batch size is 196,608 tokens. Rather than applying standard learning rate decay, we maintain an Exponential Moving Average (EMA) of the model weights with a decay rate of 0.9999. Both our final evaluations and the publicly released model weights use this EMA checkpoint.

**Infrastructure.** The parallelization framework is based on PyTorch FSDP2[58]. All models are trained in a single continuous run. We do not use intermediate checkpointing, crash recovery, or skip loss spikes.

# 5 Discussion

## 5.1 Toward decoupling knowledge and reasoning

Our results suggest a direction for partially decoupling factual coverage from reasoning computation. HRM-Text is trained on only 40B unique tokens, and explicitly knowledge-oriented sources constitute only a fraction of the task-formatted mixture. Nevertheless, the model achieves strong performance on reasoning-heavy benchmarks such as MATH and GSM8K, while retaining nontrivial performance on broader knowledge benchmarks such as MMLU. This pattern suggests that a compact recurrent model can learn useful task-execution and reasoning behavior without requiring the same degree of broad factual memorization typically associated with trillion-token pretraining.

This observation motivates future systems that separate a compact reasoning core from factual storage. In such systems, recurrent models like HRM-Text could specialize in computation, planning, and task execution, while factual breadth is supplied by curated corpora, retrieval-augmented stores, or learned memory modules. Recent conditional-memory approaches such as Engram point in a related direction: instead of forcing the Transformer backbone to simulate static pattern lookup through dense computation, they introduce scalable memory lookup as a complementary sparsity axis, freeing neural computation for global context integration and reasoning[59]. HRM-Text does not yet implement retrieval or conditional memory, but its results suggest that combining small recurrent reasoning models with external or learned knowledge stores is a promising direction for future work.

## 5.2 Adaptive computation time (ACT)

Wang et al.[4] equipped HRM with an adaptive computation module that allows simpler problems to terminate earlier, reducing computation while maintaining near-optimal performance. We do not use this component in HRM-Text in order to keep the design and training procedure simpler, but it remains a promising direction for improving both performance and computational efficiency. The current recurrent schedule provides additional effective serial depth, but it also increases inference-time computation relative to a single-pass Transformer. ACT would allow easy prompts or tokens to halt after fewer recurrent cycles while reserving the full recurrent budget for harder cases, potentially recovering a substantial portion of the inference overhead. We therefore view ACT as a natural complement to HRM-Text's recurrent-depth design: HRM supplies depth when reasoning is needed, while ACT can make that depth conditional rather than fixed.

## 5.3 PrefixLM with inference frameworks

PrefixLM can run inside standard text-generation inference frameworks such as vLLM without requiring a fundamentally different serving stack. The main requirement is custom attention-mask handling during prefilling, so that instruction tokens can attend bidirectionally while response tokens remain autoregressive.

Using a PrefixLM-style attention pattern in multi-turn chat also requires careful KV-cache logic: user tokens need full attention within each user segment, while assistant tokens must preserve causal generation. This is an engineering constraint rather than a conceptual limitation, but it should be addressed explicitly in production inference systems.

# 6 Conclusion

We introduced HRM-Text as an empirical existence proof that highly efficient pretraining is achievable. Inspired by biological multi-timescale processing, we co-designed a hierarchical recurrent architecture with a targeted task-completion objective. This demonstrates that there is at least one model family capable of reaching competitive performance without relying on the massive compute and internet-scale raw text that dominate current paradigms.

By drastically reducing the compute-to-performance ratio, this work opens significant potentials for future research. Foundational pretraining is no longer locked inside highly resourced institutions; it is now computationally accessible to small labs, academic groups, and even individuals. We hope this democratization empowers the broader community to actively explore, train, and innovate on new architectures from scratch.

# 7 Related Work

The literature on recurrent neural networks and language modelling is extensive. In this section, we discuss the most relevant papers.

## 7.1 Scaling laws and efficient pretraining

Language-model development is driven by scaling laws and compute-optimal training, which together prescribe jointly increasing parameters, data, and compute[1,2]. This underlies the dominant recipe: large decoder-only Transformers trained on massive corpora and refined via mid- and post-training[60,61,62,63]. While this scaling paradigm has produced strong models, it concentrates pretraining among compute-rich organizations, reinforcing a growing compute divide[64,65]. HRM-Text instead explores whether improved architectures, objectives, and data curation can shift the cost–performance frontier, complementing scaling laws by increasing per-token and per-FLOP efficiency.

## 7.2 Conditional sequence modeling and PrefixLM

The distinction between modeling conditional answers, $P_\theta(x_a \mid x_q)$, and full text streams, $P_\theta(x)$, predates modern LLMs. Early sequence-to-sequence models and encoder–decoder transformers explicitly model outputs conditioned on inputs[15,66,67,29]. T5[16] later unified NLP tasks as text-to-text generation, reinforcing this conditional framing. In the instruction-tuning phase of language modeling, NLP datasets are converted into instruction–response pairs, and a mask is often applied so that loss is only computed on the response tokens[12,13]. Scaling approaches like FLAN show that such task formatting improves generalization[14,68].

Decoder-only models concatenate the prompt and response into a single causal stream and predict all tokens. Although scalable, this is inefficient: the prompt is known at inference time, yet training still assigns loss to reconstruct it.

PrefixLM-style objectives bridge decoder-only models and conditional generation: prefix tokens attend bidirectionally, while outputs remain causal[17,18,16,3]. HRM-Text builds directly on this lineage by making conditional modeling the primary pretraining objective, using response-only loss and PrefixLM masking to combine encoder–decoder behavior with decoder-only simplicity.

### 7.3 Latent computation and recurrent language models

A line of work seeks to improve model capability by increasing internal computation rather than just scaling parameters or output tokens. Universal Transformers introduced recurrent depth to self-attention[33], and later recurrent or block-recurrent Transformer variants reuse parameters across steps or layers[7,6,69]. These approaches echo classic recurrent-network ideas but inherit the challenge of unstable long-range credit assignment[5,8].

Recent latent-reasoning approaches refine hidden states internally before emitting an answer[70,71]. Recurrent-depth language models such as Huginn and looped language models such as Ouro[72] scale this idea to language modeling and test-time computation[24,72]. Meanwhile, CCDD[73] establishes the connection between looped transformers and continuous diffusion language model with latent reasoning advantages. These works demonstrate that latent recurrence is a promising alternative to purely token-level reasoning, but many still rely on large token budgets, stage-wise training, or extensive test-time recurrence.

HRM-Text builds on the Hierarchical Reasoning Model, which uses a two-timescale recurrent design for symbolic reasoning[4]. Like prior work, it emphasizes richer internal computation, but it differs in that it is trained from scratch under a small token budget and uses a hierarchical dual-timescale architecture. Related work such as TRM explore even smaller recursive models[35], suggesting that hierarchy, temporal separation and recurrence can enable useful serial computation, though applying them to language remains challenging due to larger states and broader data.

### 7.4 Stable recurrent optimization

Stability is a key challenge for recurrent-depth language models. In Transformers, normalization placement trades off forward stability and gradient flow: PostNorm stabilizes activations but is harder to optimize at depth, while PreNorm improves gradients but risks residual growth and reduced expressivity[9,10]. Recurrence intensifies this issue, as repeated transformations create long products of Jacobian-like operators during backpropagation. Prior work shows exact long-horizon credit assignment is often impractical[5,11], and studies of random matrix products and neural gradients suggest deep multiplicative paths lead to heavy-tailed, lognormal-like variability[74,75,76].

HRM-Text addresses these stability issues using architecture-specific techniques: *MagicNorm* and warm-up for deep credit assignment. These design choices distinguish HRM-Text from generic looped Transformers and are crucial to making recurrent depth stable at language-model scale.

## Acknowledgements

We thank Sen Song, Jiacheng You, and Andy L. Siy for their insightful discussions.

# References


1. 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. *arXiv preprint arXiv:2001.08361*, 2020.
2. Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
3. Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. UL2: Unifying language learning paradigms. In *International Conference on Learning Representations*, 2023.
4. Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.
5. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.
6. Jishnu Ray Chowdhury and Cornelia Caragea. Investigating recurrent transformers with dynamic halt. *arXiv preprint arXiv:2402.00976*, 2024.
7. DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 33248–33261, 2022.
8. Nicolas Zucchet and Antonio Orvieto. Recurrent neural networks: Vanishing and exploding gradients are not the end of the story. In *Advances in Neural Information Processing Systems*, 2024.
9. Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pages 10524–10533. PMLR, 2020.
10. Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, 2020.
11. Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.
12. Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.
13. Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022.

14. Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22631–22648. PMLR, 2023.
15. Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.
16. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
17. Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018.
18. Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
19. Meta AI. Llama 3: State-of-the-art open weight language models. Technical report, Meta, 2024. URL https://ai.meta.com/llama/.
20. An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
21. Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.
22. Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.
23. 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, 2025.
24. Jonas Geiping, Sean Michael 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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=S3GhJooWIC.
25. Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.
26. Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.
27. Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

28. Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *Advances in Neural Information Processing Systems*, 38: 100092–100118, 2026.
29. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
30. Eugene M. Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007. doi: 10.1093/cercor/bhl152.
31. Ryunosuke Amo, Sara Matias, Akihiro Yamanaka, Kenji F Tanaka, Naoshige Uchida, and Mitsuko Watabe-Uchida. A gradual temporal shift of dopamine responses mirrors the progression of temporal difference error in machine learning. *Nature neuroscience*, 25(8): 1082–1092, 2022.
32. Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993. doi: 10.1016/0010-0277(93)90058-4.
33. Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.
34. Ibrahim Alabdulmohsin and Xiaohua Zhai. Recursive inference scaling: A winning path to scalable inference in language and multimodal systems. *Advances in Neural Information Processing Systems*, 38:109020–109049, 2026.
35. Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.
36. Yi Hu, Cai Zhou, and Muhan Zhang. What affects the effective depth of large language models? *arXiv preprint arXiv:2512.14064*, 2025.
37. Damien Sileo. tasksource: A large collection of nlp tasks with a structured dataset preprocessing framework. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, 2024.
38. HuggingFace H4. No robots. https://huggingface.co/datasets/HuggingFaceH4/no_robots, 2023. Dataset card.
39. PleIAs. Pleias/synth · datasets at hugging face, 2025. URL https://huggingface.co/datasets/PleIAs/SYNTH.
40. Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. Platypus: Quick, cheap, and powerful refinement of llms, 2023.
41. Pranjal Aggarwal, Marjan Ghazvininejad, Seungone Kim, Ilia Kulikov, Jack Lanchantin, Xian Li, Tianjian Li, Bo Liu, Graham Neubig, Anaelia Ovalle, Swarnadeep Saha, Sainbayar Sukhbaatar, Sean Welleck, Jason Weston, Chenxi Whitehouse, Adina Williams, Jing Xu, Ping Yu, Weizhe Yuan, Jingyu Zhang, and Wenting Zhao. Reasoning over mathematical objects: On-policy reward modeling and test time aggregation, 2026.
42. Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. In *International Conference on Learning Representations*, 2025.

43. Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. https://huggingface.co/datasets/AI-MO/NuminaMath-CoT, 2024.
44. Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Zhengyang Tang, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, volume 2025, pages 100540–100569, 2025.
45. David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.
46. Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.
47. Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. In *Advances in neural information processing systems*, volume 38, pages 110320–110345, 2026.
48. Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, et al. Openthoughts: Data recipes for reasoning models, 2025.
49. Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning, 2025.
50. Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Dong Wang, Ilia Kulikov, Kyunghyun Cho, Yuandong Tian, Jason Weston, et al. Naturalreasoning: Reasoning in the wild with 2.8 m challenging questions. In *Advances in Neural Information Processing Systems*, volume 38, 2026.
51. Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. In *Advances in Neural Information Processing Systems*, volume 38, pages 56596–56618, 2026.
52. Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. Openchat: Advancing open-source language models with mixed-quality data. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AOJyfhWYHf.
53. Yi Dong, Zhilin Wang, Makesh Sreedhar, Xianchao Wu, and Oleksii Kuchaiev. Steerlm: Attribute conditioned sft as an (user-steerable) alternative to rlhf. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11275–11288, 2023.
54. DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. doi: 10.48550/arXiv.2501.12948. URL https://arxiv.org/abs/2501.12948.

55. Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, et al. Scaling up models and data with t5x and seqio. *Journal of Machine Learning Research*, 24(377):1–8, 2023.
56. Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
57. Katie E Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *Forty-first International Conference on Machine Learning*, 2024.
58. Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. Torchtitan: One-stop pytorch native solution for production ready LLM pretraining. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SFN6Wm7YBI.
59. Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.
60. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in neural information processing systems*, volume 33, pages 1877–1901, 2020.
61. Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
62. Emmy Liu, Graham Neubig, and Chenyan Xiong. Midtraining bridges pretraining and posttraining distributions, 2025.
63. Kaixiang Mo, Yuxin Shi, Weiwei Weng, Zhiqiang Zhou, Shuman Liu, Haibo Zhang, and Anxiang Zeng. Mid-training of large language models: A survey, 2025.
64. Tamay Besiroglu, Sage Andrus Bergerson, Amelia Michael, Lennart Heim, Xueyun Luo, and Neil Thompson. The compute divide in machine learning: A threat to academic contribution and scrutiny? *arXiv preprint arXiv:2401.02452*, 2024.
65. Nur Ahmed and Muntasir Wahed. The de-democratization of ai: Deep learning and the compute divide in artificial intelligence research. *arXiv preprint arXiv:2010.15581*, 2020.
66. Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1179.

67. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.
68. Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25:1–53, 2024.
69. Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025.
70. Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E. Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling*, 2025.
71. Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts, 2025.
72. 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, 2025. URL https://arxiv.org/abs/2510.25741.
73. Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. In *Forty-third International Conference on Machine Learning*, 2026.
74. Boris Hanin and Mihai Nica. Products of many large random matrices and gradients in deep neural networks: B. hanin, m. nica. *Communications in Mathematical Physics*, 376(1):287–322, 2020.
75. Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner, and Daniel Soudry. Neural gradients are near-lognormal: Improved quantized and sparse training. *arXiv preprint arXiv:2006.08173*, 2020.
76. Liam Hodgkinson and Michael W. Mahoney. Multiplicative noise and heavy tails in stochastic optimization. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 4262–4272. PMLR, 2021. URL https://proceedings.mlr.press/v139/hodgkinson21a.html.
77. Tero Karras, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. Guiding a diffusion model with a bad version of itself. *Advances in Neural Information Processing Systems*, 37:52996–53021, 2024.
78. Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

# Appendix

## A FLOPs estimation

For dense models, we use the standard training-FLOPs estimate $F = 6ND$.

For recurrent models, we account separately for the forward and backward recurrent unrolls. We count $2ND$ for forward computation and $4ND$ for backward computation, then scale these terms by the number of recurrent steps included in each pass.

## B Evaluation details

Table 8: Shared evaluation configuration.

| Setting | Value |
|---|---|
| Maximum evaluation context | 3072 tokens |
| Decoding temperature | 0 |
| System prompt | None |
| Prompt contents | Original question only |
| Baseline scores | Original papers when available; otherwise rerun from open weights |

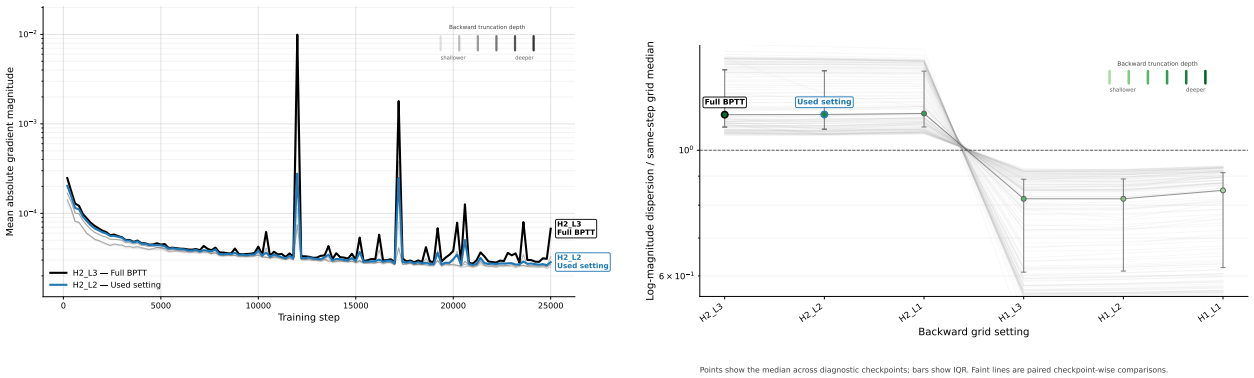
The evaluation prompt contains the original benchmark question and, when required by the benchmark protocol, the corresponding few-shot examples. Few-shot examples are added only for few-shot evaluations. We do not add an additional system prompt. Unless otherwise specified, decoding is deterministic with temperature zero and a maximum context length of 3072 tokens.

Baseline scores are taken from the original papers when those scores are available under comparable settings. When paper-reported numbers are unavailable, we evaluate the corresponding open-weight model directly. All few-shot evaluations are run with the same configuration used for HRM-Text and use the vLLM inference engine. Chain-of-thought evaluations are run with `lm_eval_harness`.

## C Stable optimization in recurrent-depth models

### C.1 Gradient Stability Under Deep BPTT in HRM

Backpropagation through time (BPTT) is the canonical mechanism for training recurrent computation graphs, yet extensive prior work has established that propagating gradients through the full unroll is often unnecessary and can be detrimental to optimization. On the other hand, truncating the backward horizon can improve practical convergence by trading exact long-range credit assignment for gradients that behave more favorably as stochastic estimators. This bias–stability tradeoff has been formalized and leveraged in the recurrent literature, motivating both principled truncation schemes and analyses of when and why truncation improves training dynamics[11]. In contrast, analogous diagnostics remain underdeveloped for HRM, or other modern looped architectures, where repeated application of a shared block induces an implicit recurrence and the effective backward depth is controlled by the number of backpropagated loop iterations.




(a) Mean absolute gradient magnitude over training.

(b) Log-magnitude dispersion.

Figure 6: (a) Full BPTT exhibits rare but substantially larger gradient-magnitude spikes compared with the truncated setting, suggesting that longer backward horizons introduce intermittent high-amplitude gradient events. (b) Values are normalized within diagnostic checkpoints to isolate the effect of backward depth from global training-time drift. Deeper H cycling increases log-magnitude dispersion

We hypothesize that the instabilities observed under deep BPTT in looped architectures are a consequence of the intrinsically multiplicative structure of gradient propagation through repeated iterations. Specifically, gradients backpropagate through products of Jacobian-like operators across loop steps, and theory for products of many random matrices predicts that the logarithm of the norm of such products is approximately Gaussian, implying lognormal-like variability in gradient magnitudes and increasing separation between typical and extreme values as backward depth grows[74]. Complementing this theoretical perspective, empirical evidence suggests that neural gradient magnitudes are often close to lognormal, consistent with multiplicative mechanisms that concentrate mass near small magnitudes while producing comparatively heavier tails[75,76].

To test these hypotheses, we perform a targeted study of gradient dynamics in HRM as we systematically increase the number of backward H and L cycles while holding the forward computation fixed. We first quantify instability using the *mean absolute gradient magnitude* and show in Figure 6a that extending the backward horizon toward full BPTT yields substantially more intermittent high-amplitude gradient events over training. We then characterize distributional heterogeneity using a complementary dispersion measure in Figure 6b, which reports *log-magnitude dispersion*, defined as $\mathrm{Std}(\log(|g| + \varepsilon))$. This measure supports the view that deeper backward cycling increases multiplicative heterogeneity in gradient magnitudes. Notably, the increase in log-magnitude dispersion is driven primarily by the H-cycle depth rather than the L-cycle depth. We therefore interpret the H dimension as the dominant contributor to gradient-magnitude spread in these experiments.

Finally, we examine whether the observed gradient spikes are consistent with a multiplicative amplification mechanism. Figure 7a shows that Jacobian growth increases with backward depth, indicating that deeper backpropagation through the recurrent computation amplifies some directions more strongly. Figure 7b provides a paired comparison between the truncated setting used for training and the full-BPTT reference at identical diagnostic checkpoints. The paired scatter shows that full BPTT is often comparable to truncation but occasionally produces much larger gradient magnitudes, consistent with the hypothesis that full backward unrolling primarily harms optimization through rare, high-amplitude tail events rather than through a uniform increase in gradient scale.

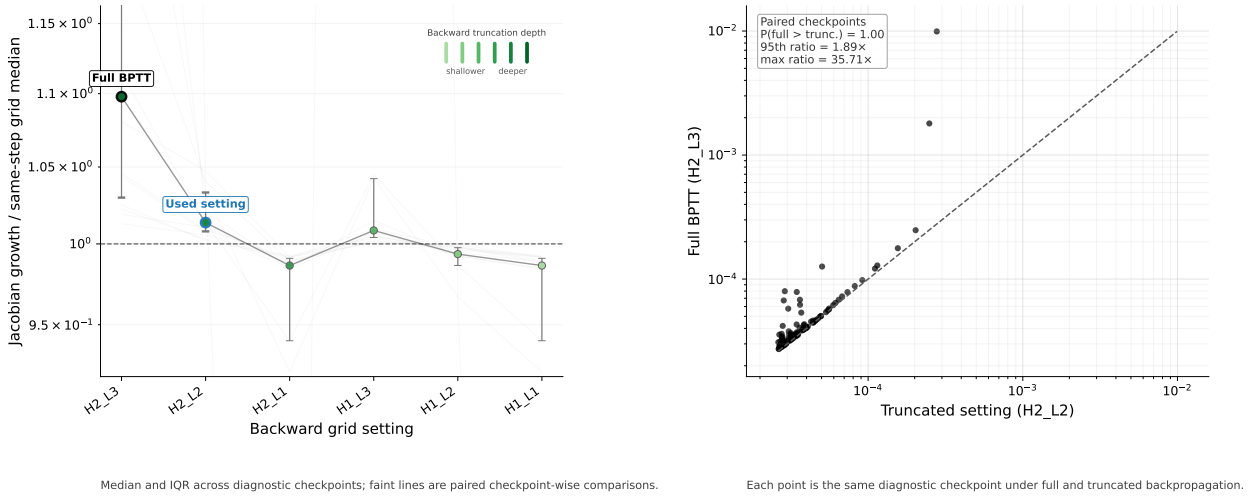



Figure 7: **Mechanistic evidence for multiplicative gradient instability.** Left: Jacobian growth increases with deeper backward cycling, consistent with stronger amplification through products of loop Jacobians. Right: paired full-vs-truncated gradient magnitudes show that full BPTT produces rare, disproportionately large gradient events at the same diagnostic checkpoints.

The truncation setting used in our experiments is the closest setting to full BPTT that remains stable during training, as illustrated in Figure 6a and Figure 7.

## C.2 Gradient stability across recurrent architectures

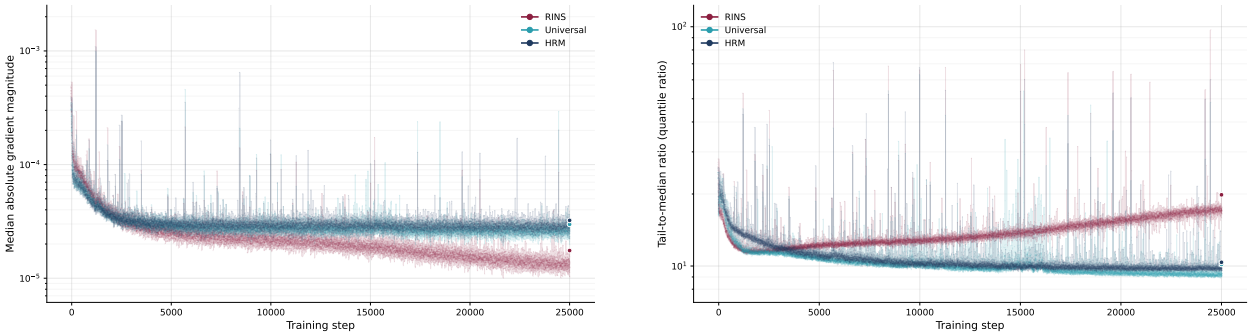
We further compare HRM against RINs and the Universal Transformer through the lens of gradient stability. Since all three architectures reuse computation over depth or recurrence, stable gradient dynamics are an important part of whether the architecture can be trained effectively, rather than merely whether it has sufficient expressive capacity. We therefore evaluate two complementary statistics across runs: the median absolute gradient magnitude and the tail-to-median ratio.

Figure 8a shows that HRM and the Universal Transformer maintain a stronger training signal as optimization progresses: their median absolute gradient magnitudes remain higher than those of RINs across training. At the same time, Figure 8b shows that this signal does not come from increasingly unstable or rare extreme updates. Instead, HRM and the Universal Transformer exhibit lower tail-to-median ratios over training, indicating that the gradient distribution becomes more even, less heavy-tailed, and less dominated by rare large updates.

This places HRM in the favorable regime of retaining useful gradient signal while avoiding the instability associated with heavy-tailed gradient dynamics. Combined with the main results, this suggests that HRM preserves the training stability of stronger recurrent-depth baselines while delivering better downstream performance.

# D Inference-time analysis

At inference time, we enable the auto-guidance[77] mechanism specifically designed for HRM, which guides itself by interpolating or extrapolating logits from various recursion depths. While having




(a) Median absolute gradient magnitude across runs.

(b) Tail-to-median gradient ratio across runs.

Figure 8: Gradient stability comparison between RINs, HRM, and the Universal Transformer. HRM maintains a strong gradient signal while exhibiting increasingly even gradient dynamics over training, matching the stability of the Universal Transformer more closely than RINs.

| Model | Guidance | MMLU | ARC-C | Hella. | Wino. | BoolQ |
|---|---|---|---|---|---|---|
| HRM 1B | w/o | 61.89 | 80.80 | 62.18 | 73.01 | 88.17 |
| HRM 1B | w/ | **62.12** (w=0.1) | 80.80 (w=0) | **62.31** (w=-0.1) | **73.17** (w=0.5) | **88.29** (w=-0.5) |

Table 9: **Inference-time auto-guidance.** We report the base performance of standard inference, and the best performance along with the corresponding guidance scale $w \in \{-0.5, -0.1, 0, 0.1, 0.5\}$.

similar motivations, auto-guidance is more efficient than classifier-free guidance (CFG)[78]: it induces zero computation overhead because the hidden representations from shallow loops are already accessible at decoding time.

In particular, suppose we have the final hidden state $h$ and another hidden state $h'$ from an earlier recurrent loop, both decoded by the LM head. Auto-guidance with guidance scale $w$ is calculated as:

$$\text{logits}_w = (1 + w) \cdot \text{logits}(h) - w \cdot \text{logits}(h')$$

$w = 0$ recovers the standard final prediction; $w > 0$ corresponds to extrapolation between the final layer and a shallower layer, treating the shallower prediction as a negative direction; and $w < 0$ corresponds to interpolation, where the model balances predictions from shallow and deep recurrent states.

Table 9 reports HRM performance with and without auto-guidance, where the guidance scale is searched over $w \in \{-0.5, -0.1, 0, 0.1, 0.5\}$. We use an HRM model with two high-level loops and interpolate or extrapolate the logits from these two $H$ modules. Because the intermediate hidden states are already available, auto-guidance introduces no additional computation. It slightly improves performance at test time, and the best guidance scale varies across benchmarks, suggesting that different tasks may benefit from different effective recurrent depths.

Auto-guidance is also closely related to adaptive computation time (ACT) and test-time scaling (TTS). When interpolation ($w < 0$) yields better results, the task may not require the full recurrent depth, suggesting that early stopping could improve efficiency. Conversely, when extrapolation

($w > 0$) performs better, the task may benefit from deeper recurrent computation and could be a candidate for adaptive test-time scaling. The results in Table 9 therefore suggest that HRM inference can support adaptive control of recurrent depth, balancing efficiency and performance at test time.