Hidden Decoding at Scale: Latent Computation Scaling for Large Language Models

WeChat AI Team

Abstract

Scaling Large Language Models (LLMs) has been driven mainly by enlarging the Transformer backbone, but for an already-strong model this requires another round of costly pretraining. We study whether an existing backbone can keep improving by allocating more computation to each token while leaving the Transformer backbone fixed. Depth-recurrent (looped) Transformers pursue this goal but are hard to scale, because looped computation does not fit naturally with the pipeline parallelism used to train the largest models. We add computation along the sequence-length dimension, where the extra computation is simply a longer input and stays compatible with standard large-model training. We propose **`Hidden Decoding`**, a sequence-length scaling method applied during continued pretraining (CPT). It expands each token into n streams with independent embedding tables and keeps the intermediate streams' key–value cache as context, so each token performs more internal computation without adding or widening Transformer layers. To keep this affordable at scale, we introduce *Stream-Factorized Attention*, in which most layers attend only within each stream and only a few layers mix across streams, reducing the attention cost from quadratic to roughly linear in n. Experiments support two scaling results. At frontier scale, we train WeLM-HD4-80B and WeLM-HD4-617B at n=4 and improve their matched non-HD baselines, making Hidden Decoding the first demonstrated sequence-length scaling method at the 100B+ MoE scale. Across expansion factors, the gains grow as n increases, showing that sequence-length expansion is a practical fixed-backbone scaling path for frontier-scale LLMs.

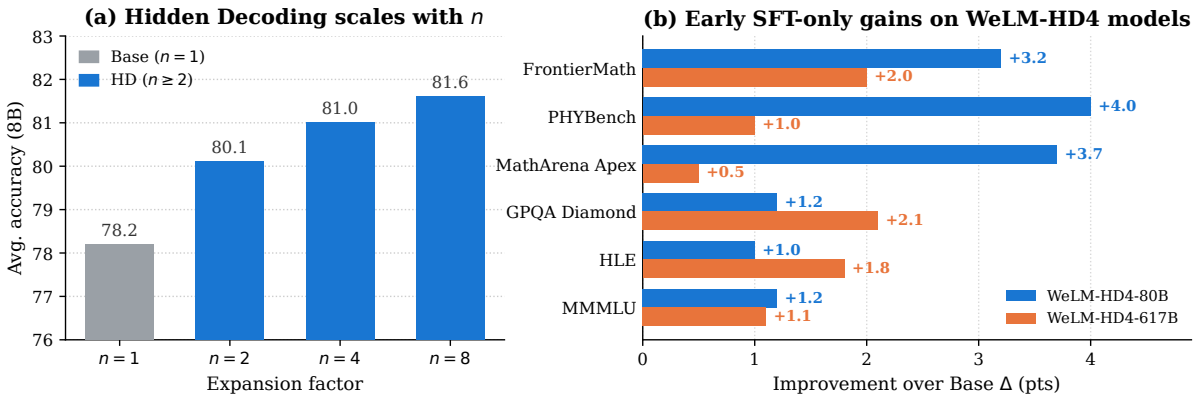

Figure 1 | **Hidden Decoding scales and improves frontier models. (a)** On the dense Qwen3-8B-Base, average accuracy grows steadily with the expansion factor n. **(b)** Improvement Δ (in points) of WeLM-HD4-80B and WeLM-HD4-617B over their matched non-HD counterparts, WeLM-80B and WeLM-617B, after the same early SFT-only post-training.

1. Introduction

Scaling has been a primary source of progress in Large Language Models (LLMs). Increasing model size, training data, and training compute has consistently led to stronger models [16, 21]. However, for already strong foundation models, further scaling the Transformer backbone is costly: it often requires another round of large-scale pretraining and increases both training and serving costs. This motivates the setting of this paper: improving an existing backbone by giving each token more internal computation while keeping the Transformer backbone fixed.

Recent reasoning models and test-time scaling studies show that, even with fixed parameters, spending more computation per problem can improve accuracy [9, 27, 29, 39]. This motivates a goal that recent latent-reasoning work has started to pursue: moving part of the extra thinking computation from visible tokens into the model's internal computation.

A prominent direction toward this goal is recurrent-depth or looped Transformers, which add computation by reusing the same Transformer blocks several times on each token [10, 31, 38, 54]. This is a direct way to increase per-token computation with a fixed set of Transformer weights. However, it is hard to scale. Training the largest MoE models relies on pipeline parallelism [8, 17, 28, 42], which assumes each input passes through the model's stages only once. A looped model breaks this assumption, because it feeds the same hidden states back through the same stages on every iteration, which stalls the pipeline and leaves GPUs idle. Consistent with this, looped models have stayed small: the largest is the 40B dense LoopCoder, trained without pipeline parallelism [51], while other looped language models remain at a few billion parameters [10, 54]. Looping therefore still lacks a practical way to scale to very large models while keeping GPUs efficiently utilized.

A key observation is that sequence-dimension computation fits large-scale training better than depth reuse. Along the sequence dimension, the expansion simply amounts to feeding a longer input, which is naturally compatible with the standard optimizations used for large-model training. The general paradigm is to expand each input token into several streams along the sequence, process the expanded sequence in a single forward pass, and apply the next-token loss only to the final stream, so the intermediate streams add per-token computation without adding new Transformer layers or widening existing ones. One instantiation is the Parallel Hidden Decoding Transformer (PHD) [49], which repeats each token but, to keep inference cheap, makes the repeated-token KV transient: intermediate streams can help within the current token, but they are not retained as independent KV context for later tokens. The capability-scaling setting therefore needs persistent intermediate-stream states that remain available across positions during CPT.

In this work, we propose a sequence-length scaling method named **`Hidden Decoding`**, which improves the capability of frontier-scale LLMs through continued pretraining (CPT). Two design choices make the expanded streams useful for capability scaling. To give different streams distinct initial states, we replicate the vocabulary embedding table into n per-stream tables, expanding each token x_i into streams $(E_1(x_i), E_2(x_i), \ldots, E_n(x_i))$; through CPT, these tables learn diverse initial representations of the same token. To let intermediate computation persist across positions, we keep the KV of the intermediate streams, so the computation accumulated across streams stays available as context for later tokens.

Retaining the KV of all streams makes intermediate computation available to later tokens, but it also makes the attention cost grow as $O(n^2L^2)$, which becomes infeasible to train for large models with long sequences. We therefore introduce *Stream-Factorized Attention*. The key idea is to limit cross-stream attention to a subset of layers: most layers attend only within each

stream, and the streams exchange information only at the remaining layers, which follow the base model's attention pattern (sliding-window, or full when the base model uses full attention). Because dense attention over the full nL sequence is thus avoided at most layers, the added attention cost stays roughly linear rather than quadratic in n. This efficiency is what makes Hidden Decoding practical to train at the scale of MoE models with over 100B parameters, a regime not reached by prior looped or length-scaling methods.

Experiments directly test the two scaling claims. To test whether Hidden Decoding scales to the 100B+ MoE regime, we train WeLM-HD4-80B and WeLM-HD4-617B at n=4. WeLM-HD4-80B improves all nine shared benchmarks over WeLM-80B, with large gains on SciCode (45.8 → 50.0) and PHYBench (69.8 → 73.8). WeLM-HD4-617B also improves all nine shared benchmarks over WeLM-617B, including GPQA Diamond (89.1 → 91.2), HLE (33.6 → 35.4), and FrontierMath (49.0 → 51.0). These results use an early SFT-only post-training recipe: each matched pair of non-HD and HD models uses the same SFT training recipe, with a short supervised fine-tuning schedule and no reinforcement learning. We use them as controlled comparisons; mature WeLM release scores are outside the scope of this paper. To test whether the expansion factor itself provides scaling, we increase n from 2 to 8 in the 80B progressive-expansion study. MMLU improves monotonically (85.0 → 86.7 → 87.5), and Pile-test BPB falls from 0.386 to 0.378. Together, these results establish sequence-length scaling as a practical fixed-backbone scaling path for frontier-scale LLMs.

Main Contributions

- **Frontier-Scale Fixed-Backbone Scaling.** To our knowledge, we are the first to demonstrate sequence-length scaling at the 100B+ MoE scale through continued pretraining (CPT). This yields two key models, WeLM-HD4-80B and WeLM-HD4-617B, in a regime not reached by prior depth-recurrent (looped) or length-scaling approaches.
- **Efficient Sequence-Length Expansion.** To make the expanded sequence trainable at frontier scale, we introduce *Stream-Factorized Attention*: most layers attend only within each stream, while a small subset mixes streams. This keeps the attention cost near-linear in n; on WeLM-HD4-80B and WeLM-HD4-617B, the 4× expanded sequence costs only 5.1× and 4.4× per batch, respectively, far below the dense-attention 16× baseline.
- **Expansion-Factor Scaling.** We show that increasing the expansion factor improves language-modeling loss and downstream accuracy, validating n as a practical scaling knob for a fixed Transformer backbone.

2. Method

`Hidden Decoding` is a sequence-length scaling method that increases the computation each token receives without enlarging the Transformer backbone. Each input token is represented in n parallel streams inside the Transformer, so a length-L sequence is processed as a length-nL sequence in a single forward pass, giving each token n internal computation steps before its prediction. Figure 2 contrasts Hidden Decoding with depth-recurrent (looped) computation: Hidden Decoding places the extra computation along the sequence dimension, while looped computation reuses the backbone along the depth dimension. This section formalizes the multi-stream expansion and training objective (§2.1), introduces *Stream-Factorized Attention* to keep the expansion affordable at scale (§2.2), and describes how to grow the expansion factor progressively from a converged checkpoint (§2.3).

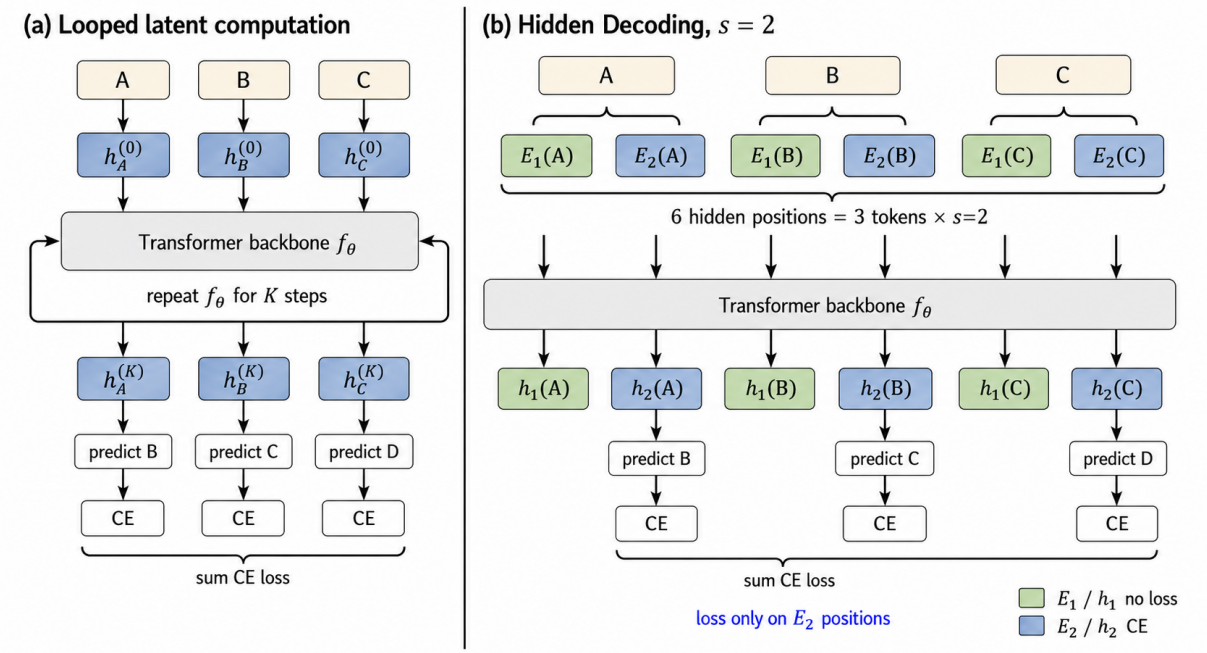

Figure 2 | **Looped latent computation vs. Hidden Decoding.** **(a)** Depth-recurrent (looped) computation reuses the same backbone f_θ for K steps on each token's hidden state before predicting the next token. **(b)** Hidden Decoding expands each token into n streams with independent embedding tables $E_1, \ldots, E_n$, interleaving them into a length-nL sequence that is processed in a single pass through the same backbone. The next-token cross-entropy is applied only at the final-stream positions (E_n/h_n); the earlier streams ($E_1, \ldots, E_{n-1}$) receive no loss and act as latent computation states. The panel shows a two-stream example.

Setup and notation. A standard Transformer language model maps each token x_i through a single vocabulary embedding table E to form the input sequence $(E(x_1), \ldots, E(x_L))$; it adds positions with Rotary Positional Embeddings (RoPE) [40], applies causal attention, and predicts the next token from each position's hidden state through a shared head $g_\theta(\cdot)$. Hidden Decoding leaves this path unchanged and changes only how each token is embedded into the sequence.

2.1. Multi-Stream Token Expansion

The core construction of Hidden Decoding is to replace the single embedding table of a standard model with n independent embedding tables, which we call streams, and to interleave their outputs into one longer input sequence.

Expanded sequence. Given an input sequence $X = (x_1, \ldots, x_L)$ and n embedding tables $E_1, \ldots, E_n$, we form an expanded sequence S of length nL by placing the stream-k representation of token x_i at physical position $t = (i-1)n + k$:

$$S_t = E_k(x_i), \qquad 1 \le i \le L, \quad 1 \le k \le n. \tag{1}$$

For $n = 2$, this yields $S = \big(E_1(x_1), E_2(x_1), E_1(x_2), E_2(x_2), \ldots\big)$. The expanded sequence is fed to the same Transformer under standard causal attention, with contiguous RoPE positions $0, 1, \ldots, nL-1$. The expansion factor n directly controls how much extra per-token computation is added.

Training objective. Only the final stream of each token, $E_n(x_i)$, is supervised: from its hidden state we predict the next token x_{i+1} through the shared language-modeling head g_θ, while the first $n-1$ streams receive no direct loss. Letting h_t denote the hidden state at physical position t, the objective is the next-token cross-entropy applied only at the final-stream positions $t = in$:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{L} \log g_\theta\big(x_{i+1} \mid h_{in}\big). \tag{2}$$

Because attention is causal, the final stream $E_n(x_i)$ attends to all earlier streams of the same token and of all preceding tokens. The first $n-1$ streams therefore act as intermediate computation states that progressively refine the representation before the final, prediction-bearing stream. Supervising only the final stream and giving each stream its own embedding table give the intermediate streams this latent-computation role; supervising all streams, or summing their outputs before prediction, degrades performance in the supervision-design ablation in §5.1.

2.2. Stream-Factorized Attention

Running dense attention over the nL expanded positions costs $O(n^2L^2)$, so the training cost grows sharply for large LLMs with long sequences. *Stream-Factorized Attention* reduces this by making most layers attend only within a stream and confining across-stream mixing to a subset of layers.

Stream and token index. For a position $t \in \{1, \ldots, nL\}$ in the expanded sequence, let its token and stream indices be

$$i(t) = \lceil t/n \rceil, \qquad s(t) = \big((t-1) \bmod n\big) + 1, \tag{3}$$

so that $S_t = E_{s(t)}(x_{i(t)})$. Consecutive blocks of n positions correspond to consecutive tokens, with the n streams ordered inside each block.

Intra-stream and cross-stream layers. Using the indices in Eq. 3, an intra-stream layer restricts attention to earlier positions of the same stream,

$$M^{\text{intra}}_{t,t'} = \mathbf{1}[\, t' \le t \ \wedge\ s(t') = s(t)\,], \tag{4}$$

which costs $O(nL^2)$ instead of the $O(n^2L^2)$ of a dense layer; a cross-stream layer also attends across streams, following the model's usual attention pattern. Figure 3 illustrates the three resulting mask types.

Design. Most layers are intra-stream, and cross-stream attention is enabled at only a subset of layers. We add no new attention layers for this: we reuse the base model's own attention layers as the cross-stream layers. When the base model contains sliding-window attention, those layers serve as the cross-stream layers, so cross-stream attention is local; otherwise the cross-stream layers are full-attention layers. Because most layers are intra-stream, the added attention cost stays close to linear in n rather than quadratic (§3); the number and placement of cross-stream layers is a design choice we study in §4. This layout also keeps CPT closer to the checkpoint before HD expansion: most layers preserve the base model's single-stream causal path, while only a few layers introduce cross-stream perturbations; Appendix H gives a separate 617B-scale startup-loss check for this effect.

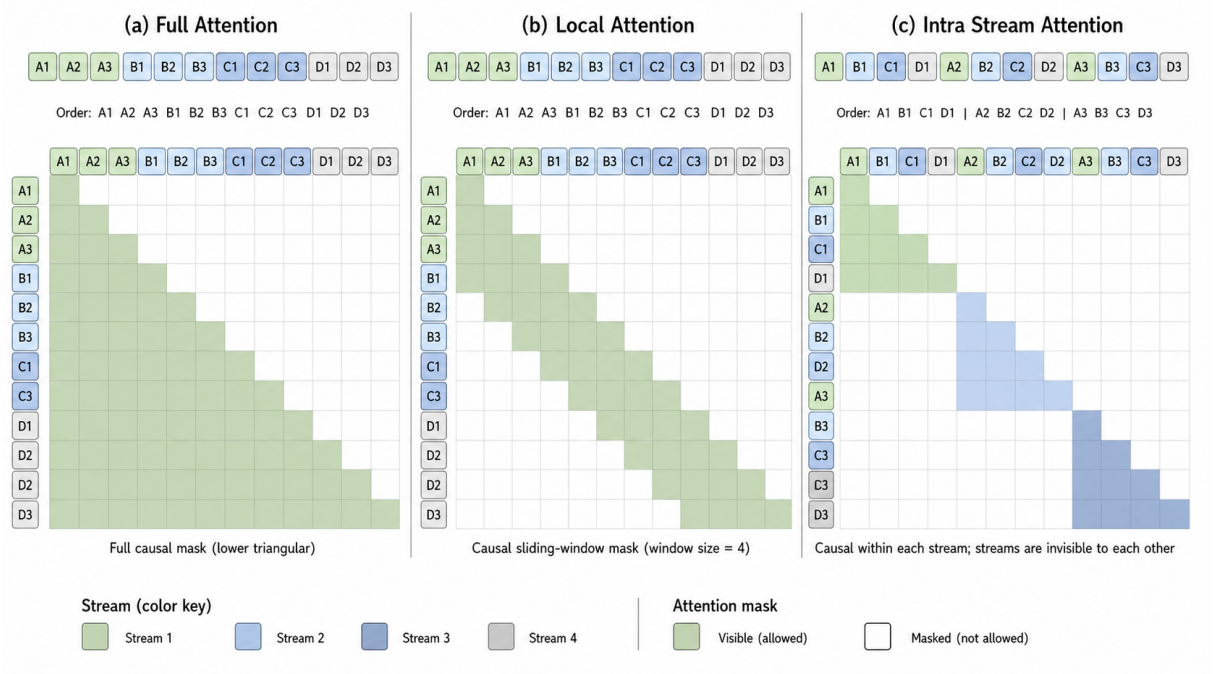

Figure 3 | **Attention masks in Stream-Factorized Attention.** Each panel shows which positions a query (row) may attend to (colored) in the expanded sequence. **(a)** A full cross-stream layer uses the standard causal (lower-triangular) mask. **(b)** A local cross-stream layer uses a causal sliding-window mask. **(c)** An intra-stream layer is causal within each stream, so the streams are invisible to one another. Stream-Factorized Attention makes most layers intra-stream (c) and enables cross-stream mixing (a or b) at only a few layers.

2.3. Progressive Expansion

A large expansion factor n can be trained directly, but progressively growing it provides a better initialization for the many embedding streams and saves compute. We therefore grow the factor progressively—$1 \to 2 \to 4 \to 8$—initializing each factor from the converged checkpoint of the previous one. Concretely, we use Cyclic Replication Initialization: when doubling from n to $2n$, the new streams copy the existing tables,

$$E_{n+k} \leftarrow E_k, \qquad 1 \le k \le n, \tag{5}$$

so that the expanded model initially reproduces the behavior of the smaller one and then adapts smoothly. For example, $n{=}2 \to 4$ initializes the tables as (E_1, E_2, E_1, E_2), and $n{=}4 \to 8$ as $(E_1, E_2, E_3, E_4, E_1, E_2, E_3, E_4)$. In practice, we introduce successive expansions at scheduled token counts during continued pretraining, letting each factor converge before the next expansion.

3. Computational Cost

Hidden Decoding adds computation to every token, so its training and serving costs must be measured. We separate two kinds of cost: the cost inherent to the method, which holds on any backbone, and an extra saving specific to our WeLM backbone. We report the training measurements in §4.2 and the serving measurements in §4.5.

3.1. Training Cost

Expanding each token into n streams turns a length-L sequence into length nL. Non-attention computation scales roughly with the number of positions, but dense attention over the expanded sequence would scale as $O(n^2L^2)$, or $n^2\times$ the attention cost of the unexpanded sequence. Stream-Factorized Attention (§2.2) avoids this dense-attention blow-up by keeping most layers intra-stream, so the attention cost stays close to linear in n. This near-linear attention design is inherent to the method and holds on any backbone.

Our WeLM backbones allow one further, architecture-specific saving through their KV-mirror design [47]. In a standard layer, the keys and values come from that layer's own input hidden states; under KV-mirror, each later layer instead computes them from an earlier layer's hidden states, using its own projection weights. The layers are paired in a U-shape, with the last third mirroring the first third. Because a mirrored layer's keys and values depend only on the early layer's hidden states, our training framework computes them at the early layers, and the mirrored layers reuse them directly.

KV-mirror lets Hidden Decoding skip work in mirrored layers. Only the final stream is supervised (§2.1), so the intermediate streams matter only through the keys and values they add as context. In the mirrored layers this context is already fixed by the early layers, and the intermediate streams carry no loss, so these layers need not process the intermediate streams at all. **We therefore run only the final stream through the mirrored layers, while the early layers still process all n streams.** In our 80B and 617B models the last third of the layers are mirrored, so the intermediate streams are skipped across that final third. This saving is specific to WeLM; it comes on top of the near-linear cost that Stream-Factorized Attention already provides on any backbone. As an implementation check, this KV-mirror optimization reduces per-batch time from 15 s to 12 s in an 80B 32k training setting, a 20% reduction (1.25× speedup). We use this number to quantify the WeLM-specific training-system saving used by WeLM-HD4-80B and WeLM-HD4-617B; the main cost comparison in §4.2 reports end-to-end HD training time under the long-context settings. The net effect is that training cost grows near-linearly with the number of streams n. We measure this at both scales in §4.2 (Figure 4a).

3.2. Inference Cost

A key advantage of Hidden Decoding over looped models is that its extra computation runs in parallel. A looped model adds computation by repeating the backbone step by step, and each step waits for the previous one, so the added cost cannot be hidden. Hidden Decoding spreads its computation across the n streams, which are processed together in a single forward pass. At large batch sizes, decoding is compute-bound, so this extra computation lowers throughput. At small batch sizes, decoding is bound by memory bandwidth and leaves much of the compute idle; the parallel streams can use that idle compute, so the added cost stays small. The one remaining overhead in this regime is the larger KV cache, which Stream-Factorized Attention keeps modest by reading the full expanded KV at only a few layers. **The throughput penalty is therefore smallest in the latency-sensitive, short-input/small-batch regime and grows with larger batches or longer inputs**, as we measure in §4.5.

For WeLM models, KV-mirror can also reduce decoding-side computation by skipping mirrored-layer work for intermediate streams. The serving benchmark in §4.5 uses the current benchmark implementation without this additional KV-mirror decoding optimization, because integrating it into the decoding kernel requires separate engineering. The reported throughput therefore does not include this WeLM-specific speedup.

4. Experiments

We use the experiments to test three parts of Hidden Decoding (HD). The frontier-scale MoE study checks whether HD improves the largest models we deploy, producing WeLM-HD4-80B and WeLM-HD4-617B at acceptable training cost (§4.2). The expansion-factor study checks whether adding more streams behaves like a scaling axis (§4.3). The ablations and throughput measurement check which attention choices and serving costs matter in practice (§4.4, §4.5).

4.1. Setup

Models. We apply HD on top of strong pre-trained checkpoints through continued pretraining (CPT), without enlarging the Transformer backbone. Our main HD models are WeLM-HD4-80B and WeLM-HD4-617B, obtained by expanding WeLM MoE checkpoints at 80B (3B activated) and 617B (23B activated) total parameters to n=4. Their matched non-HD counterparts are WeLM-80B and WeLM-617B. When reporting base-model evaluations before post-training, we append `-Base`, giving WeLM-80B-Base vs. WeLM-HD4-80B-Base and WeLM-617B-Base vs. WeLM-HD4-617B-Base. We use smaller 21B (0.7B activated) and 6B (0.6B activated) MoE models for ablations, and the dense Qwen3-8B-Base [50][12] for progressive expansion and probes.

Only the embedding tables grow with the expansion factor n, while the active Transformer parameters per token stay unchanged. For example, expanding to n=4 increases the stored embedding parameters from 26.9B to 107.4B for WeLM-HD4-617B and from 1.2B to 3.1B on Qwen3-8B-Base (reaching 5.6B at n=8). These tables are sparse lookups: each position reads a single embedding row, and the tables do not enter the attention or feed-forward matrix multiplications that set the compute cost. We therefore report the embedding growth separately and compare matched non-HD and HD models as a fixed-Transformer-backbone, fixed-active-parameter test.

Configurations. The large MoE models are expanded to n=4. The dense model is expanded progressively to $n \in \{2, 4, 8\}$ for the expansion-factor study. Following §2.2, most layers are intra-stream and cross-stream mixing is placed on a subset of layers. For the 80B model (49 layers), we use 6 full, 20 intra-stream, and 23 sliding-window layers. For the 617B model (94 layers), we use 25 full and 69 intra-stream layers. The base models' native context length is 256k for 80B and 32k for 617B. RoPE and context-extension settings are held fixed within each matched comparison between non-HD and HD models, including the shared early SFT recipe in Appendix D. Appendix B gives the backbone and HD configuration details for the two main WeLM models.

Continued pretraining. We introduce Hidden Decoding during continued pretraining. We start from a baseline checkpoint and continue training with the expansion enabled, using the same data and schedule as the baseline except for the 617B long-context-stage difference in Table 1. The expansion stays on from that point onward: it carries from the 32k continuation stage into a later 256k long-context stage, and then through the early SFT-only post-training recipe. The non-HD baseline is trained in the same way with the expansion turned off, so the two runs differ only in Hidden Decoding. For the 80B comparison, the two runs see identical data at every stage. For the 617B comparison, the HD long-context stage uses fewer tokens than the non-HD model, so its gains are a conservative lower bound.

Table 1 reports the training-token budget behind this comparison.

[1] `https://github.com/Tencent/Sequential-Hidden-Decoding`.
[2] `https://huggingface.co/collections/tencent/sequential-hidden-decoding`.

Table 1 | **Continued-pretraining token budgets for non-HD base paths and HD4 continuation windows.** Non-HD rows count the full base-model path. HD4 rows count only the training after the HD start point.

Scale	Run	HD start tokens	8k pretrain	32k continuation	256k continuation	Total
80B	WeLM-80B-Base	–	17.81T	2.01T	0.57T	20.39T
80B	WeLM-HD4-80B-Base	19.32T	0	0.50T	0.57T	1.07T
617B	WeLM-617B-Base	–	14.25T	2.20T	0.61T	17.06T
617B	WeLM-HD4-617B-Base	15.86T	0	0.59T	0.30T	0.90T

For 80B, Hidden Decoding starts after 19.32T tokens on the non-HD training path, and WeLM-HD4-80B-Base then trains on the same remaining 0.50T 32k tokens and 0.57T 256k tokens as the non-HD continuation. For 617B, Hidden Decoding starts after 15.86T tokens on the non-HD training path. The HD4 run matches the remaining 0.59T 32k continuation, but uses a shorter 256k continuation (0.30T vs. 0.61T). Relative to the full non-HD base paths, the HD4-only token budgets are about 5.3% at both scales, so the higher per-token cost is concentrated late in training.

Post-training scope. The post-training used for the early post-training tables is intentionally lightweight: it is an early supervised fine-tuning run and does not include reinforcement learning. We use this stage to test whether the gains from Hidden Decoding persist under the same downstream adaptation recipe for the non-HD and HD models. Appendix D gives the SFT recipe. We report these scores as controlled early-SFT comparisons, with mature WeLM release scores left outside the scope of this paper.

Evaluation. We use two evaluation suites. For early post-training results (Tables 2 and 11), we evaluate on recent hard benchmarks covering mathematics and science (HMMT and MathArena Apex [2], IMO-AnswerBench [24], FrontierMath [11], PolyMath [46], GPQA [37], PHYBench [35], CritPt [53]), knowledge and reasoning (HLE [33], MMMLU [30], AA-OmniScience [19], AA-LCR [1], ARC-AGI-2 [5]), code (SciCode [44], LiveCodeBench [20]), instruction following (IF-Bench [34]), and agentic tasks (Terminal-Bench 2 [43], τ^2-Bench [3], GDPval [32]).

For scaling and ablation studies (§4.3, §4.4) and pretraining-stage results (Appendix I), we use a standard suite: MMLU [14], MMLU-Pro [45], CMMLU [22], C-Eval [18], SuperGPQA [25], ARC-C [6], HellaSwag [52], BBH [41], GSM8K [7], MATH [15], SimpleQA [48], Chinese SimpleQA [13], HumanEval+/MBPP+ [23], MultiPL-E [4], and CRUXEval [12]. Tables 8 and 9 give the per-benchmark settings.

4.2. Frontier-Scale MoE Results

To test whether Hidden Decoding strengthens the largest models we deploy, we compare two forms of each WeLM MoE checkpoint that share everything except Hidden Decoding: WeLM-80B vs. WeLM-HD4-80B, and WeLM-617B vs. WeLM-HD4-617B. The HD models add Hidden Decoding during continued pretraining (Table 1); both forms are then adapted with the same early SFT-only post-training recipe, so the comparison isolates our method. Table 2 evaluates both scales on the same nine hard benchmarks.

Table 2 | **Early SFT♣ results for WeLM-HD4-80B and WeLM-HD4-617B.** We compare each HD4 model with its matched non-HD counterpart: WeLM-80B vs. WeLM-HD4-80B and WeLM-617B vs. WeLM-HD4-617B. HD4 denotes Hidden Decoding with n=4; active Transformer parameters per token are unchanged (3B for 80B, 23B for 617B). Both forms use the same early SFT-only post-training recipe, with a short SFT schedule and no RL. Kimi K2.6 [26] is included as a mainstream frontier-model reference for the absolute score scale; its model card reports 1T total parameters and 32B activated parameters. ‡HMMT is the weighted average over the Feb. 2025, Nov. 2025, and Feb. 2026 sets.

	80B MoE		617B MoE		External
Benchmark	**WeLM 80B**	**WeLM-HD4 80B**	**WeLM 617B**	**WeLM-HD4 617B**	**Kimi K2.6 1T-A32B**
GPQA Diamond	87.6	88.8	89.1	91.2	90.4
HLE	27.4	28.4	33.6	35.4	36.9
MMMLU	84.4	85.6	86.4	87.5	88.0
FrontierMath⋆	45.8	49.0	49.0	51.0	53.2
PHYBench	69.8	73.8	75.3	76.3	74.0
MathArena Apex	16.4	20.1	24.2	24.7	23.8
HMMT‡	93.3	94.1	96.0	96.2	96.0
IMO-AnswerBench	85.0	85.3	87.5	88.5	91.5
SciCode	45.8	50.0	51.4	52.1	50.7

♣We use early SFT here to keep the non-HD and HD models matched in post-training data and configuration. A public report gives benchmark results for a newer non-HD WeLM MoE after larger-scale, higher-quality post-training: https://welm.weixin.qq.com/en/posts/hidden_decoding_at_scale/#conclusion-and-future-work.
⋆FrontierMath uses the public Tiers 1–4 sample problems: https://epoch.ai/frontiermath/tiers-1-4/benchmark-problems.

Hidden Decoding improves every benchmark at both scales. The largest gains appear on hard math and science tasks: WeLM-HD4-80B improves SciCode by +4.2, PHYBench by +4.0, and FrontierMath by +3.2; WeLM-HD4-617B improves GPQA by +2.1 and HLE by +1.8. We also include Kimi K2.6 (1T total, 32B activated) as a mainstream frontier-model reference for absolute score scale: WeLM-HD4-617B is higher on GPQA Diamond, PHYBench, MathArena Apex, HMMT, and SciCode, while Kimi K2.6 is higher on HLE, MMMLU, FrontierMath, and IMO-AnswerBench. On the broader 80B suite in Appendix E, the largest gains shift toward agentic and long-horizon tasks, including Terminal-Bench 2 (44.9 → 58.4) and ARC-AGI-2 (6.9 → 11.6). **These results show that Hidden Decoding can be scaled to frontier-size MoE models and improve them without enlarging the Transformer backbone.** This is the scaling regime where looped models have been hard to apply, because repeated depth passes fit poorly with the pipeline parallelism used to train large MoEs.

Training cost measurement. To verify the training-cost analysis in §3, we measure per-batch training time on the WeLM backbones relative to the unexpanded baseline. Figure 4(a) shows that a 4× longer effective sequence costs 5.1× on 80B (256k → 1M) and 4.4× on 617B (32k → 128k). These costs stay close to the 4× linear reference and far below the 16× dense-attention cost. **This near-linear cost makes WeLM-HD4-617B trainable, while dense attention over the expanded sequence would be infeasible.** Using full attention on only a few layers is a deliberate trade-off; we measure its effect on accuracy with an ablation on a smaller model in §4.4.

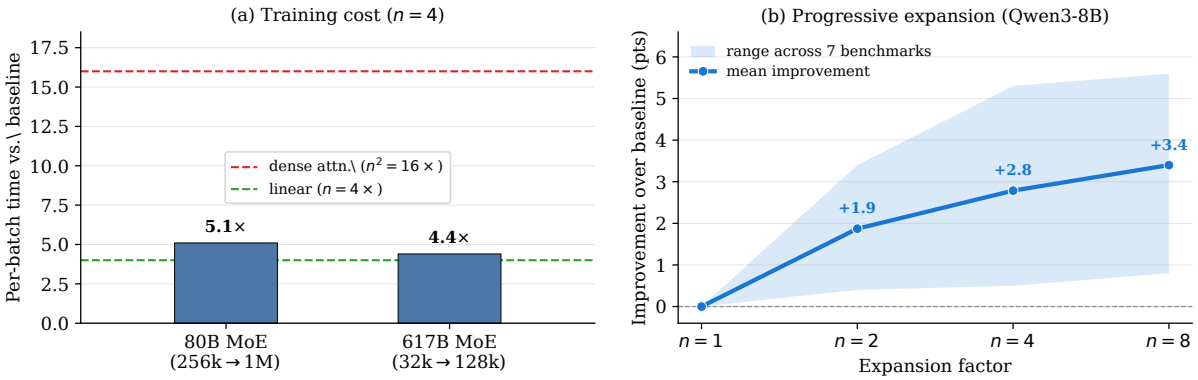

Figure 4 | **(a) Training cost of HD (n=4):** per-batch time relative to the unexpanded baseline, staying near the linear (n=4×) reference and far below dense attention (n^2=16×). **(b) Progressive expansion on the dense Qwen3-8B-Base:** the mean improvement over the unexpanded baseline grows steadily with the expansion factor n; the shaded band shows the range across the seven benchmarks.

Table 3 | **Base-model reference.** The WeLM columns use our base-model protocol before instruction tuning or RL. External numbers are copied from the Qwen3.5 report's base-model table and provide absolute-score context; the matched WeLM columns give the controlled comparison between non-HD and HD models.

Benchmark	**WeLM 617B-Base**	**WeLM-HD4 617B-Base**	DS-V3.2 671B-A37B	K2 1T-A32B	Qwen3.5 397B-A17B
MMLU	89.85	90.22	88.11	87.38	88.61
MMLU-Pro	72.26	73.92	62.82	67.64	76.01
MMLU-Redux	89.81	90.94	87.29	86.65	89.09
SuperGPQA	55.33	57.05	43.46	44.86	57.96
C-Eval	91.90	92.50	90.48	91.82	91.82
BBH	92.41	93.07	86.03	89.11	90.98
KoRBench	52.08	54.88	54.00	53.84	54.08
CRUX-Input	86.00	91.75	63.25	70.50	71.13
CRUX-Output	89.38	91.13	73.88	77.13	82.38

External-model reference. To place the base-model strength of WeLM-HD4-617B-Base in context, we compare it with the base-model table reported in the Qwen3.5 report [36]. Table 3 keeps the overlapping benchmarks from that report and our base-model evaluation. We include both WeLM-617B-Base and WeLM-HD4-617B-Base, evaluated with our base-model protocol before instruction tuning or reinforcement learning. Because model families and evaluation harnesses still differ, the controlled evidence remains the matched comparison between non-HD and HD models under the same backbone, data, and training configuration. The WeLM baseline scores are obtained by evaluating the matched non-HD Base checkpoint with the same harness as the HD Base checkpoint.

Table 4 | **Attention-composition ablation (21B MoE).** We compare a no-expansion baseline with three HD variants that differ only in the number of full cross-stream layers. Header parentheses give the number of full-attention layers (out of 27). SF with 4 full layers nearly matches all-full, while SF with 1 full layer still improves the baseline.

		Hidden Decoding		
Benchmark	Baseline	SF (1)	SF (4)	all-full (27)
MMLU	74.1	75.7	76.4	76.3
MMLU-Pro	47.5	50.4	49.8	50.7
CMMLU	77.9	79.2	79.8	79.6
C-Eval	78.6	78.9	79.2	79.7
ARC-C	89.5	89.8	90.7	90.9
SuperGPQA	34.1	35.4	35.7	36.0
BBH	67.0	71.2	72.0	72.8
GSM8K	83.4	86.2	86.7	86.2
MATH	45.2	49.6	49.3	50.3
SimpleQA	3.7	4.0	3.6	4.1
AA-OmniScience	12.7	12.4	14.2	14.1
HumanEval+	37.4	37.6	39.2	40.5
MBPP+	55.2	56.1	57.8	59.1
Average	54.33	55.88	56.49	**56.95**

4.3. Expansion-Factor Scaling

To test whether the expansion factor is a usable scaling knob, we expand the dense Qwen3-8B-Base step by step to $n \in \{2, 4, 8\}$ while keeping the Transformer backbone fixed. Figure 4(b) reports improvement over the unexpanded baseline. The mean improvement rises almost steadily with n, and every benchmark gains; the largest improvements reach +5.6 on HellaSwag and +5.1 on BBH and MATH.

The same trend appears at MoE scale. Expanding the 80B MoE raises MMLU from 85.1 at n=1 to 87.5 at n=8 and lowers Pile-test BPB from 0.386 to 0.378 (full per-factor results in Appendix G). **Within a fixed Transformer backbone, the expansion factor acts as a reliable scaling knob: increasing it improves accuracy and language-modeling loss.**

4.4. Attention Composition Ablation

To test how much full cross-stream mixing is required for accuracy, we ablate the attention composition inside Stream-Factorized Attention. Each layer mixes streams in one of three ways: full (across all streams over the whole sequence), local (across streams within a sliding window), or intra-stream (no cross-stream mixing). Full mixing is the expensive case.

We run this ablation on the 21B MoE, which has 27 layers and whose base model already interleaves full-attention and sliding-window layers. The reference is a standard no-expansion baseline. The three Hidden Decoding variants differ only in how many layers use full cross-stream mixing: 1, 4, or all 27, with the remaining layers split between local and intra-stream mixing (exact layouts in Appendix F). Table 4 reports the evaluated benchmarks and their average.

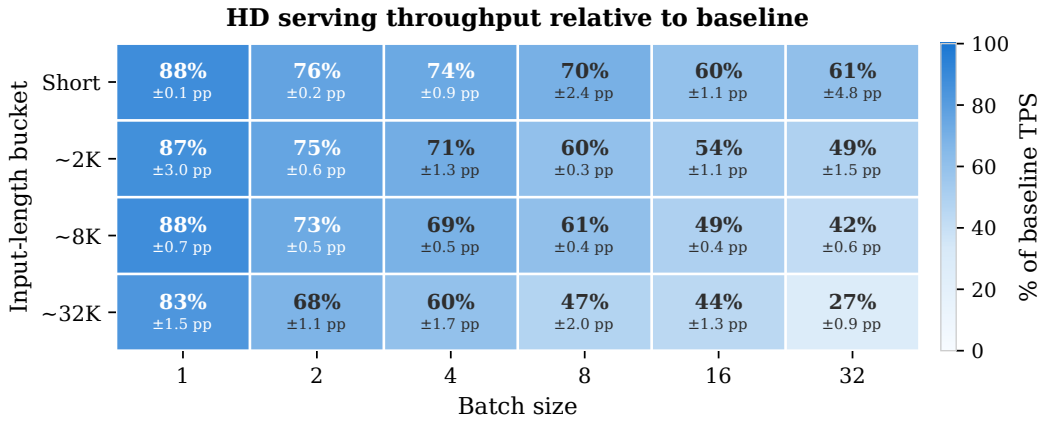

Figure 5 | **Serving throughput matrix for WeLM-HD4-80B.** Each cell reports mean decoding-only completion TPS for Hidden Decoding over five runs as a percentage of the matched WeLM-80B baseline with stream expansion disabled under the same input-length bucket and batch size; prefill latency is excluded. Small text shows the five-run sample standard deviation in percentage points. Both models run on the same 8 H20 GPUs. The benchmark uses a 30-second warmup, a 45-second measurement window, and maximum output length 8192.

Every Hidden Decoding model beats the no-expansion baseline on average, with gains from +1.55 to +2.62. The all-full model is best, but the cheaper 4-full variant is close (+2.16), and the 1-full variant still improves the baseline (+1.55). **A few full layers are enough: the remaining cross-stream mixing can be local or skipped, which keeps Stream-Factorized Attention cheap while preserving most of the accuracy gain.**

4.5. Inference Throughput

To measure serving cost under a matched setting, we vary the two factors that most directly affect decoding throughput: input length and batch size. For WeLM-HD4-80B, we serve each request bucket twice on the same 8 H20 GPUs: once with Hidden Decoding enabled, and once on the matched WeLM-80B baseline with stream expansion disabled. The benchmark reports decoding-only completion TPS after excluding prefill latency, with maximum output length 8192, a 30-second warmup, and a 45-second measurement window. Each cell in Figure 5 reports the mean Hidden Decoding completion throughput over five runs as a percentage of the matched baseline under the same input-length bucket and batch size; the small label gives the run-to-run standard deviation in percentage points.

The measurements identify a practical serving window. At batch size 1, Hidden Decoding keeps 83–88% of baseline throughput across all measured input buckets. For the short, 2k, and 8k buckets, it keeps 69–88% through batch size 4, so latency-oriented small-batch serving preserves most baseline decode throughput. The cost becomes substantial in long-context, high-batch decoding: at batch size 16, the ratio falls to 60% for short inputs, 54% for the 2k bucket, 49% for the 8k bucket, and 44% for the 32k bucket; the largest measured setting, 32k input at batch size 32, keeps 27%. These matched measurements support the cost model in §3.2: Hidden Decoding is practical for interactive or moderate-batch workloads, while long inputs and large batches expose the extra per-token computation more directly.

Table 5 | **Supervision-design ablation (6B MoE).** All variants use the same backbone and training setup. At the same expansion factor n=2, applying loss to every stream (all-token loss) or summing stream outputs (sum) underperforms supervising only the final stream (HD). Loss: lower is better.

Method	**Loss** ↓	MMLU	ARC-C	C-Eval	CMMLU
Baseline	1.908	53.4	68.7	58.4	61.2
All-token loss (n=2)	1.880	55.5	65.8	59.1	63.2
Sum (n=2)	1.877	55.6	71.9	61.0	63.3
HD (n=2)	1.874	56.5	74.3	61.2	63.7
HD (n=3)	**1.857**	**58.5**	**75.7**	**62.5**	**65.9**

5. Ablations and Probes of Intermediate Streams

Hidden Decoding relies on an unusual choice: only the final stream is trained to predict the next token, while the earlier streams receive no direct loss (§2.1). The matched results in §4.2 show that Hidden Decoding improves accuracy. The accuracy gain alone leaves open what role the earlier streams play: a gain over a no-expansion baseline could come from simpler effects such as a longer expanded sequence, more prediction targets, or explicit aggregation of stream outputs. To test these possibilities, we use ablations and probes. The training-objective ablation compares final-stream supervision with all-token loss and output summation (§5.1). The KV-retention analysis tests whether per-stream KV is useful, and stream probes inspect hidden-state separation and final-stream attention (§5.2). The LM-head probes provide a vocabulary-space view of the intermediate stream states (§5.3).

5.1. Training Objective Ablation

To separate the effect of final-stream supervision from simpler alternatives, we train objective variants on the same 6B MoE backbone and training setup. The comparison asks whether the gain comes from leaving intermediate streams without a direct loss, or whether similar gains appear when every stream receives a prediction target or when stream outputs are explicitly aggregated. The baseline has no stream expansion. The expanded variants include all-token loss, which applies the next-token loss to every stream; sum, which adds the stream outputs together before prediction; and Hidden Decoding, which supervises only the final stream. The main comparison uses the same expansion factor n=2 for all expanded variants; we also report HD at n=3 to show whether the preferred objective continues to improve with more streams.

Table 5 reports the result. On language modeling, all expanded variants improve over the no-expansion baseline, but HD gives the lowest loss at the same expansion factor: 1.874 at n=2, compared with 1.880 for all-token loss and 1.877 for sum. The downstream results follow the same pattern. HD at n=2 is best on all four reported benchmarks, with the clearest separation on ARC-C (74.3 vs. 71.9 for sum and 65.8 for all-token loss). Increasing HD to n=3 further lowers loss to 1.857 and improves all four scores. These results favor final-stream supervision over both extra direct supervision on intermediate streams and explicit summation of stream outputs.

Table 6 | **Small KV-retention ablation (21B MoE, HD n=2).** This is a qualitative check of whether stream-specific KV matters. Accuracy columns are few-shot `loss_acc` in percent (higher is better).

Full layers	KV type	Few-shot accuracy						Avg
		ARC	EEGPQA	MMLU	MMLU-Pro	QA-MMLU	SuperGPQA	
1	Per-stream	91.50	54.17	74.62	48.83	80.90	35.36	64.23
	Shared	91.01	53.19	74.24	46.60	81.07	34.63	63.46
4	Per-stream	92.32	54.95	75.24	47.82	82.11	35.50	64.66
	Shared	91.28	53.32	74.41	47.90	81.15	35.35	63.90

5.2. KV Retention Ablation and Stream Probes

Hidden Decoding normally retains a separate KV cache for each stream, so later positions can use the computation produced by earlier intermediate streams. To test whether this retained per-stream KV is useful, we compare two Hidden Decoding variants that keep the same backbone, training setup, expansion factor, and Stream-Factorized Attention layout. The reference setting is the normal per-stream KV design. The controlled variant is shared KV, a PHD-like control [49]: it keeps the final-stream prediction objective and separate stream trajectories, but removes independently retained stream KV by using one shared KV cache across streams in intra-stream layers. If the intermediate-stream KV were redundant, replacing per-stream KV with this PHD-like control should have little systematic effect.

We run this as a small supporting ablation on the 21B MoE at n=2. This setting is intentionally limited, so we use it as a qualitative check of direction. We report two Stream-Factorized Attention layouts, SF(1 full) and SF(4 full), which keep one or four full cross-stream layers. In each layout, the only intended difference between the two rows is whether KV is retained separately per stream or shared across streams.

Because this is a small n=2 comparison, Table 6 should be read qualitatively. Shared KV lowers the average in both layouts: 64.23 → 63.46 with one full cross-stream layer and 64.66 → 63.90 with four full cross-stream layers. The per-task columns are noisy, as expected in this small comparison, but the average moves in the same direction in both layouts. These consistent average drops indicate that preserving separate KV states for different streams improves accuracy in this setting.

The KV ablation establishes that retaining per-stream KV affects accuracy. To connect this accuracy effect to internal computation, we inspect what the streams contain and whether the final stream reads them. We analyze the trained dense Qwen3-8B-Base Hidden Decoding model at n=8, where E7 is the final prediction stream and E0–E6 are intermediate streams. Figure 6 gives two probes: hidden-state similarity measures whether streams form different internal states, and attention affinity measures whether E7 reads other streams.

Panel (a) shows that the streams form different internal states. Same-token streams start highly aligned, separate through the middle layers, and move closer again near the output: the mean cosine drops from 0.987 at the input to 0.637 in the middle layers, then rises to 0.783 at the final layer. The curve remains above the different-token baseline, which separates stream-specific structure from generic hidden-state anisotropy. Panel (b) shows that these states are connected to the prediction stream. E7 assigns substantial attention to other streams, especially

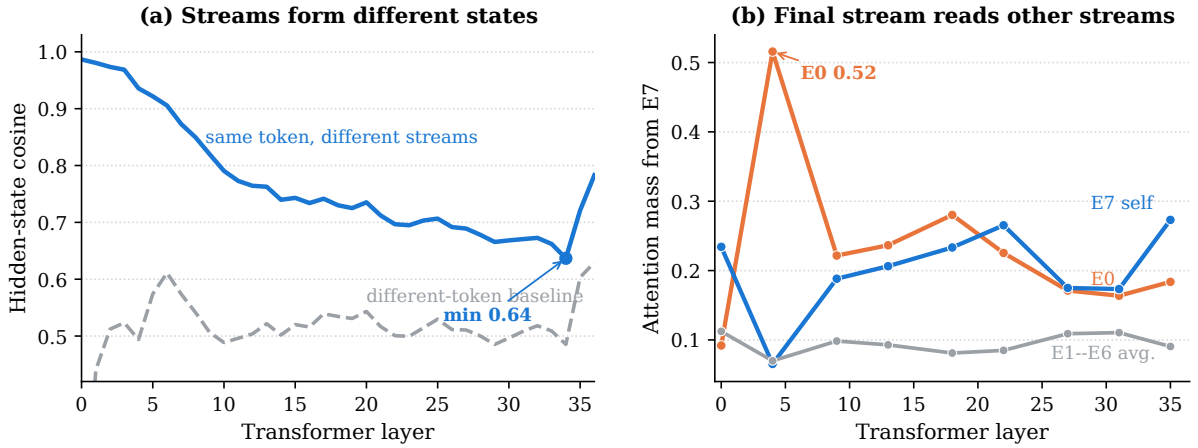

Figure 6 | **Intermediate streams form states that the final stream can read (Qwen3-8B-Base + Hidden Decoding, n=8). (a)** Same-token streams separate inside the Transformer and partially move closer near the final layer; the dashed curve is a different-token baseline. **(b)** The final stream E7 assigns substantial attention to other streams, especially E0, showing a read path from intermediate streams to the prediction stream.

E0, whose affinity peaks at 0.52 and is comparable to or larger than self-attention in several sampled layers. **Together, the KV ablation and the probes support the same interpretation: intermediate streams carry distinct states, and retaining their KV gives later computation access to those states.**

5.3. LM-Head Probes in Vocabulary Space

To see how intermediate stream states appear at the output vocabulary level, we apply the shared LM head to each stream hidden state and inspect the resulting next-token distribution. This probe shows what token candidates are visible at different stream positions.

Figure 7 gives two aggregate views. Panel (a) uses the dense Qwen3-8B-Base Hidden Decoding model at n=8 and measures how often each stream's probed top-1 token differs from the final stream E7. Intermediate probes often differ from E7, with the largest difference rate around 63%. Panel (b) compares mean probe entropy across HD models with n=2, 4, 8. The final stream has the lowest entropy in each setting; at n=8, E7 has entropy 2.09 bits, while several intermediate streams remain above 3 bits. **Together, the top-1 differences and higher entropy suggest that intermediate streams keep a broader token-level candidate space, while the final stream consolidates it into the prediction.**

Figure 8 provides qualitative examples of the same pattern. Their role is illustrative: in these prompts, early or middle probes may favor template, structural, or associated tokens, such as "a", "located", or "Apollo", while later probes move closer to the final stream's token. These examples show the broader-candidate-space interpretation at the prompt level.

Overall, the ablations and probes support the same interpretation. The supervision-design comparison shows that final-stream supervision works better than training the intermediate streams to predict or summing their outputs. The KV and attention analyses show that stream-specific KV is useful, the streams separate inside the Transformer, and the final stream reads other streams. The LM-head probes provide an interpretability view: intermediate streams preserve a broader set of token alternatives before the final stream settles on a prediction.

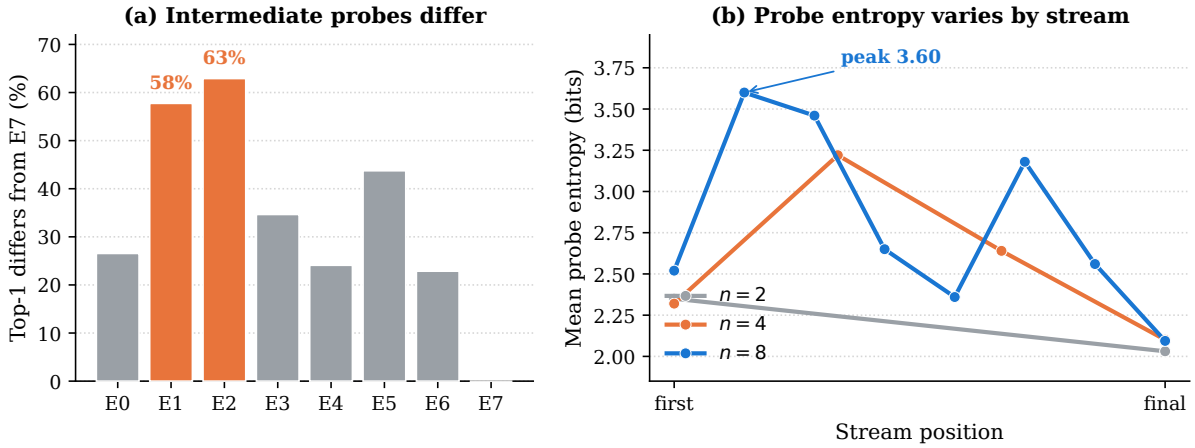

Figure 7 | **LM-head probes show broader intermediate token distributions (Qwen3-8B-Base + Hidden Decoding).** We apply the shared LM head to each stream hidden state and inspect the resulting probed next-token distribution. **(a)** In the n=8 model, intermediate probes often have a top-1 token different from the final stream E7. **(b)** Intermediate probes have higher entropy than the final stream, indicating broader token-level uncertainty before the final prediction.

	E0	E1	E2	E3	E4	E5	E6	E7 (final)
The capital of France is _	**a** 0.51	**is** 0.74	**a** 0.38	**a** 0.23	**a** 0.23	**_** 0.16	**Paris** 0.13	**Paris** 0.49
The capital of Japan is _	**located** 0.20	**and** 0.60	**located** 0.41	**located** 0.25	**Tokyo** 0.22	**_____** 0.59	**Tokyo** 0.28	**Tokyo** 0.69
...first person to walk on the _	**Apollo** 0.82	**Moon** 0.36	**True** 0.59	**Moon** 0.54	**bi** 0.17	**_____** 0.82	**_____** 0.47	**moon** 0.65
All that glitters is not _	**necessarily** 0.62	**gold** 0.48	**.** 0.41	**always** 0.35	**gold** 0.40	**gold** 0.57	**gold** 0.75	**gold** 0.97
import numpy as _	**↵** 0.34	**np** 0.15	**↵** 0.41	**np** 1.00	**np** 1.00	**np** 1.00	**np** 1.00	**np** 1.00

Figure 8 | **Illustrative stream-probe predictions (Qwen3-8B-Base + Hidden Decoding, n=8).** For prompts with a clear answer, the figure shows each stream probe's top-1 token and its probability. Green marks probes whose top-1 matches the final stream; grey marks probes that differ. These examples serve as qualitative illustrations of the aggregate differences in Figure 7.

Together, these results support the role of earlier streams as latent computation states.

6. Conclusion

We introduced **Hidden Decoding**, a sequence-length scaling method for improving a fixed Transformer backbone through continued pretraining. The core idea is to give each token more internal computation by expanding it into n streams, while keeping the Transformer layers unchanged. Hidden Decoding supervises only the final stream and retains the intermediate streams' KV so their computation remains available to later tokens. Stream-Factorized Attention makes this expansion trainable by keeping most layers intra-stream and limiting cross-stream mixing to a subset of layers. The ablations support these design choices: final-stream supervision outperforms direct losses on every stream or explicit summation of stream outputs, and retained

per-stream KV improves accuracy in the controlled small-model checks.

The experiments show that this scaling route is useful and trainable at frontier scale. At n=4, Hidden Decoding produces WeLM-HD4-80B and WeLM-HD4-617B and improves their matched non-HD counterparts under the same early SFT-only post-training setup. The training-time measurements explain why this scale is practical: the 4× expanded sequence costs 5.1× on 80B and 4.4× on 617B, close to linear and far below dense attention over the expanded sequence. The expansion-factor result shows that increasing n improves both the dense 8B model and the 80B MoE, with 80B MMLU rising from 85.1 to 87.5 and Pile-test BPB falling from 0.386 to 0.378.

Together, these results establish sequence-length expansion as a practical fixed-backbone scaling path for frontier-scale LLMs. It does not require training a larger Transformer backbone, and it avoids the looped execution pattern that conflicts with pipeline-parallel large-model training. Instead, the extra computation appears as a longer sequence, which fits the same engineering stack used to train very large MoE models. Hidden Decoding therefore provides an effective and engineering-realistic way to keep improving strong LLMs when further backbone scaling is costly.

References

[1] Artificial Analysis. Artificial analysis long context reasoning (AA-LCR). Online evaluation, 2025. URL https://artificialanalysis.ai/evaluations/artificial-analysis-long-context-reasoning.

[2] M. Balunović, J. Dekoninck, I. Petrov, N. Jovanović, and M. Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions, 2025. URL https://arxiv.org/abs/2505.23281.

[3] V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

[4] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, A. Guha, M. Greenberg, and A. Jangda. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. arXiv preprint arXiv:2208.08227, 2022. URL https://arxiv.org/abs/2208.08227.

[5] F. Chollet, M. Knoop, G. Kamradt, B. Landers, and H. Pinkard. ARC-AGI-2: A new challenge for frontier AI reasoning systems, 2025. URL https://arxiv.org/abs/2505.11831.

[6] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018.

[7] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

[8] DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

[9] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025. URL https://arxiv.org/abs/2501.12948.

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

[11] E. Glazer, E. Erdil, T. Besiroglu, et al. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI, 2024. URL https://arxiv.org/abs/2411.04872.

[12] A. Gu, B. Rozière, H. Leather, A. Solar-Lezama, G. Synnaeve, and S. I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. arXiv preprint arXiv:2401.03065, 2024. URL https://arxiv.org/abs/2401.03065.

[13] Y. He, S. Li, J. Liu, Y. Tan, W. Wang, H. Huang, X. Bu, H. Guo, C. Hu, B. Zheng, et al. Chinese simpleqa: A chinese factuality evaluation for large language models. arXiv preprint arXiv:2411.07140, 2024. URL https://arxiv.org/abs/2411.07140.

[14] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

[15] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

[16] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556, 2022. URL https://arxiv.org/abs/2203.15556.

[17] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Advances in Neural Information Processing Systems, volume 32, pages 103–112, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html.

[18] Y. Huang, Y. Bai, Z. Zhu, J. Zhang, J. Zhang, T. Su, J. Liu, C. Lv, Y. Zhang, J. Lei, Y. Fu, M. Sun, and J. He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. arXiv preprint arXiv:2305.08322, 2023. URL https://arxiv.org/abs/2305.08322.

[19] D. Jackson, W. Keating, G. Cameron, and M. Hill-Smith. Aa-omniscience: Evaluating cross-domain knowledge reliability in large language models. arXiv preprint arXiv:2511.13029, 2025. URL https://arxiv.org/abs/2511.13029.

[20] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

[21] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

[22] H. Li, Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan, and T. Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. arXiv preprint arXiv:2306.09212, 2023. URL https://arxiv.org/abs/2306.09212.

[23] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems, 36, 2023.

[24] T. Luong, D. Hwang, others, T. H. Trinh, and Q. V. Le. Towards robust mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.01846.

[25] M-A-P Team, X. Du, Y. Yao, K. Ma, B. Wang, T. Zheng, K. Zhu, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. arXiv preprint arXiv:2502.14739, 2025. URL https://arxiv.org/abs/2502.14739.

[26] Moonshot AI. Kimi k2.6. Hugging Face model card, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.6.

[27] N. Muennighoff, Z. Yang, W. Shi, X. L. Li, F.-F. Li, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling. arXiv preprint arXiv:2501.19393, 2025. URL https://arxiv.org/abs/2501.19393.

[28] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia. Efficient large-scale language model training on GPU clusters using megatron-LM. In International Conference for High Performance Computing, Networking, Storage and Analysis, page 58. ACM, 2021. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

[29] OpenAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, 2024. Accessed: 2026-06-07.

[30] OpenAI. MMMLU: Multilingual massive multitask language understanding. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/openai/MMMLU.

[31] T. Park, Y. Lee, D. Kim, and H. Bae. Loopus: Recasting pretrained llms into looped latent refinement models. arXiv preprint arXiv:2605.11011, 2026. URL https://arxiv.org/abs/2605.11011.

[32] T. Patwardhan, R. Dias, E. Proehl, et al. GDPval: Evaluating AI model performance on real-world economically valuable tasks, 2025. URL https://arxiv.org/abs/2510.04374.

[33] L. Phan, A. Gatti, et al. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

[34] V. Pyatkin, S. Malik, V. Graf, H. Ivison, S. Huang, P. Dasigi, N. Lambert, and H. Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://arxiv.org/abs/2507.02833.

[35] S. Qiu, S. Guo, Z.-Y. Song, et al. PHYBench: Holistic evaluation of physical perception and reasoning in large language models, 2025. URL https://arxiv.org/abs/2504.16074.

[36] Qwen Team. Qwen3.5. Qwen Blog, 2026. URL https://qwen.ai/blog?id=qwen3.5.

[37] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling, 2024.

[38] N. Saunshi, N. Dikkala, Z. Li, S. Kumar, and S. J. Reddi. Reasoning with latent thoughts: On the power of looped transformers, 2025. URL https://arxiv.org/abs/2502.17416.

[39] C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. arXiv preprint arXiv:2408.03314, 2024. URL https://arxiv.org/abs/2408.03314.

[40] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 568:127063, 2024.

[41] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint arXiv:2210.09261, 2022. URL https://arxiv.org/abs/2210.09261.

[42] K. Team. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

[43] Terminal-Bench Team (Stanford and Laude Institute). Terminal-Bench: A benchmark for AI agents in terminal environments. Online leaderboard, 2025. URL https://www.tbench.ai/leaderboard/terminal-bench/2.0.

[44] M. Tian, L. Gao, S. D. Zhang, et al. SciCode: A research coding benchmark curated by scientists, 2024. URL https://arxiv.org/abs/2407.13168.

[45] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024. URL https://arxiv.org/abs/2406.01574.

[46] Y. Wang, P. Zhang, others, J. Lin, and J. Zhou. PolyMath: Evaluating mathematical reasoning in multilingual contexts, 2025. URL https://arxiv.org/abs/2504.18428.

[47] WeChat AI. Building effective sparse moe models with moderate resources. WeChat AI blog, 2026. URL https://welm.weixin.qq.com/posts/building-effective-sparse-moe-models-with-moderate-resources/.

[48] J. Wei, N. Karina, H. W. Chung, Y. J. Jiao, S. Papay, A. Glaese, J. Schulman, and W. Fedus. Measuring short-form factuality in large language models. arXiv preprint arXiv:2411.04368, 2024. URL https://arxiv.org/abs/2411.04368.

[49] B. Wu, S. Yan, S. Zhang, J. Lu, Y. Zeng, Y. Wang, and X. Zhou. Efficient pretraining length scaling, 2025. URL https://arxiv.org/abs/2504.14992.

[50] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[51] J. Yang, W. Zhang, S. Guo, Z. Ye, L. Jing, S. Liu, Y. Li, J. Wu, C. Liu, X. Ma, Y. Song, S. Wu, Y. Li, L. Liao, T. Zheng, Z. Huang, Z. Huang, C. Liu, Y. Xing, R. Li, Q. Cai, H. Yan, S. Wang, S. Li, J. K. Liu, A. Huang, Y. Kang, J. Zhang, C. Hao, H. Wang, W. Gu, R. Tao, M. Tang, P. Wu, J. Wang, X. Liu, W. Lv, and B. Dai. Iquest-coder-v1 technical report, 2026. URL https://arxiv.org/abs/2603.16733.

[52] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? arXiv preprint arXiv:1905.07830, 2019.

[53] M. Zhu, M. Tian, X. Yang, et al. Probing the critical point (CritPt) of AI reasoning: a frontier physics research benchmark, 2025. URL https://arxiv.org/abs/2509.26574.

[54] R.-J. Zhu, Z. Wang, K. Hua, T. Zhang, Z. Li, H. Que, B. Wei, Z. Wen, F. Yin, H. Xing, et al. Scaling latent reasoning via looped language models. arXiv preprint arXiv:2510.25741, 2025. URL https://arxiv.org/abs/2510.25741.

A. Contributors

Contributors are listed alphabetically by English name.

Aiwei Liu	Cheng Shi	Chuhan Wu
Ci Lei	Di Lu	Donald He
Fan Zhang	Fanhao Kong	Feifei Zhang
Guan Wang	Haicheng Wang	Haoyu Liu
Houjin Yu	Jiachen Ding	Jiayi Feng
Jie Zhou	Jijun Chi	Jindi Shi
Jing Lei	Junjie Zhang	Laiyi Li
Le Tian	Linhao Zhang	Miao Fan
Sijun Zhang	Wei Jia	Weiwei Shi
Wenhan Li	Wentao Zhao	Wenteng Liang
Xiao Zhou	Xiaojin Zhou	Xihuai Wang
Xinyu Gao	Xuanliang Wang	Xuyang Ao
Yang Yu	Yangxiu You	Yinuo Zhao
Yufei Kuang	Yufei Wang	Yuan Liu (刘)
Yuan Liu (柳)	Yuwen Chen	Zhencong Tian
Zhongyin Zhao	Zilin Yu	Zitao Wang

B. Main WeLM Model Configurations

To make the main WeLM setting reproducible, Table 7 lists the backbone and Hidden Decoding configuration used for the 80B and 617B comparisons in §4.2. The non-HD and HD forms share the same Transformer backbone. Hidden Decoding changes the stream expansion and Stream-Factorized Attention layout, while leaving the Transformer widths and layer counts unchanged.

Table 7 | **Main WeLM backbone and HD configuration.** Activated parameters are per token. "Q/KV heads" reports query heads and grouped key–value heads. "Experts/top-k" includes one shared expert. "KV mirror" is the number of mirrored late layers; "SFA" reports intra/local/full layers.

Backbone	MoE params	Activated params	Layers	Hidden size	Q/KV heads	Head dim.	Experts / top-k	KV mirror	SFA intra/local/full
WeLM-80B	80B	3B	49	2048	24/2	256	512+1 / 10	16	20/23/6
WeLM-617B	617B	23B	94	4096	96/8	128	512+1 / 10	30	69/0/25

C. Evaluation Configurations

To make the reported scores reproducible, we separate the evaluation settings into two suites. Table 8 gives the standard suite used for scaling studies, ablations, and base-model evaluations. Table 9 gives the early post-training suite used for the main WeLM results in Table 2 and the additional 80B results in Table 11.

Table 8 reports the few-shot count, chain-of-thought (CoT) setting, and scoring method for

Table 8 | **Per-benchmark evaluation configurations.** Execution-based code benchmarks are scored at pass@1 with greedy decoding.

Benchmark	Shots	CoT	Scoring
Knowledge & multiple-choice			
MMLU	5	No	Accuracy
MMLU-Pro	5	No	Accuracy
CMMLU	5	No	Accuracy
C-Eval	5	No	Accuracy
ARC-Challenge	25	No	Accuracy
HellaSwag	10	No	Accuracy
EE-GPQA	5	No	Custom evaluator
SuperGPQA	5	No	Accuracy (A–K)
Reasoning & math			
BBH	3	Yes	EM / Accuracy
GSM8K	4	Yes	Numeric match
MATH	4	Yes	MATHEvaluator v2
MATH Modern CoT	4	Yes	MATHEvaluator v2
MATH LLM Judge	4	Yes	LLM-as-Judge
Factual QA			
SimpleQA	5	No	Substring match
Chinese SimpleQA	10	No	LLM Judge
AA-OmniScience	10	No	LLM Judge
Code			
CRUXEval	1 / 2	Yes	Execution (pass@1)
EvalPlus	1	No	Execution (pass@1)
MultiPL-E	1	No	Execution (pass@1)

each benchmark in the standard suite. For SuperGPQA, EE-GPQA, Chinese SimpleQA, and AA-OmniScience, the few-shot exemplars are pre-formatted as a fixed prefix in the data rather than retrieved dynamically; the same exemplars are therefore shared across all examples of those benchmarks. The three MATH variants share the same dataset and differ only in scoring: rule-based equivalence (v2), the same scorer with extra Chinese stopping tokens (modern CoT), and an LLM judge. This lets us separate scoring effects from model effects.

The standard suite spans four families. Knowledge and multiple-choice benchmarks are scored by exact-match accuracy on the predicted option letter, without chain-of-thought. Reasoning and math benchmarks use chain-of-thought prompting and then extract the final answer. Factual-QA benchmarks ask for short answers, scored either by normalized substring matching (SimpleQA) or by an LLM judge (Chinese SimpleQA, AA-OmniScience). Code benchmarks execute the generated programs against test cases and report pass@1 under greedy decoding. The LLM-judge benchmarks use a fixed judge model and temperature throughout, so scores stay comparable across our runs.

Table 9 lists the settings for the early post-training suite. Each question is run N times and the per-question scores are averaged; N is as large as 32 for the small, high-variance math sets (MathArena Apex, FrontierMath). Generation is free-form with a generous token budget (up to

Table 9 | **Evaluation settings for the early post-training suite (Tables 2 and 11).** N is the number of independent runs per question; scores are averaged over the N runs. LLM-judge benchmarks use a fixed judge model and temperature. ‡HMMT is the weighted average over the Feb. 2025, Nov. 2025, and Feb. 2026 sets. †External evaluations.

Benchmark	N	**Scoring**
GPQA Diamond	4	Accuracy (multiple choice)
HMMT‡	4	Weighted average; LLM judge (answer equiv.)
IMO-AnswerBench	1	LLM judge (answer equiv.)
MathArena Apex	32	LLM judge (HLE-style)
FrontierMath	32	LLM judge (HLE-style)
PolyMath	1	LLM judge (HLE-style)
PHYBench	4	LLM judge (HLE-style)
CritPt	8	Two-call, CritPt API
HLE	1	LLM judge (HLE-style)
MMMLU	1	LLM judge (HLE-style)
AA-OmniScience	2	LLM judge (4-way)
AA-LCR	3	LLM judge (equality)
SciCode	1	Execution (sub-step rate)
LiveCodeBench v6	2	Execution (pass@1)
IFBench	2	Programmatic checks (pass@1)
ARC-AGI-2	1	Exact grid match (pass@2)
Terminal-Bench 2	3	Agent + verifier (resolved rate)
GDPval†	–	Win rate
τ^2-Bench†	–	Reward

160k tokens per response), so the models can use long chains of thought. The scoring styles are as follows.

Most math, science, and knowledge benchmarks are open-ended: the model writes a full solution, and an LLM judge compares its final answer to the gold answer for equivalence, following the protocol of Humanity's Last Exam [33] (we refer to this as HLE-style judging). This covers FrontierMath, MathArena Apex, PHYBench, PolyMath, MMMLU, and HLE itself; the math benchmarks use Qwen3.5-397B-A17B as a fixed judge model. GPQA Diamond is the exception—it is multiple-choice, scored by whether the selected option matches the correct label. AA-OmniScience uses a four-way judge (correct / incorrect / partial / not-attempted) and reports accuracy, and AA-LCR uses a general equality judge over long-document questions.

The remaining benchmarks are scored by execution or by task-specific verifiers rather than a judge. SciCode generates code for each sub-step of a problem and runs it against reference assertions, scoring the fraction of sub-steps that pass; LiveCodeBench v6 and IFBench run the generated programs (or instruction-checking code) and report pass@1. ARC-AGI-2 parses the predicted grid from the response and requires an exact match, reporting task-level pass@2. Terminal-Bench 2 is agentic: for each task the model drives an agent inside a containerized environment, and an environment verifier returns a reward, which we report as the resolved rate. CritPt makes two model calls per question—first solving the physics problem, then filling in a code template—and submits the result to the CritPt scoring API. Across all LLM-judge benchmarks the judge model and temperature are fixed throughout, so scores stay comparable

Table 10 | **Early SFT hyperparameters for the matched WeLM comparisons.** The same recipe is used for each matched pair of non-HD and HD models in Table 2; no reinforcement learning stage is applied.

Item	Setting
Optimizer	Adam + Muon
Adam parameters	β_1=0.9, β_2=0.95, ε=10^{-8}
Muon parameters	Momentum 0.95, Nesterov enabled, 5 Newton–Schulz steps
Learning rate	2.0×10^{-5}
Schedule	Cosine decay to 0, with 56 warmup iterations and 1125 decay iterations
Regularization	Weight decay 0.1; gradient clipping 1.0
Precision	bfloat16
Batching	Global token batch 4,194,304; micro-batch token budget 262,144
Context extension	YaRN scaling factor 8 from a 32k base context; rotary base 500,000
Adaptation scope	LoRA disabled; supervised fine-tuning only, with no RL

across our runs.

D. Early SFT Hyperparameters

To make the early post-training comparison in Table 2 reproducible, we use the same SFT hyperparameters for each matched pair of non-HD and HD models. This supervised-only stage tests whether the gains from continued pretraining persist after the same lightweight adaptation step.

E. Additional Benchmarks for WeLM-HD4-80B

To check whether the 80B gains extend beyond the nine benchmarks shared across scales in Table 2, we evaluate WeLM-HD4-80B on a broader suite covering agentic, code, instruction-following, and long-context tasks. Table 11 compares WeLM-80B and WeLM-HD4-80B under the same early SFT-only post-training setup as §4.2. WeLM-HD4-80B improves eight of the ten benchmarks, with the largest gains on Terminal-Bench 2 (44.9 → 58.4) and ARC-AGI-2 (6.9 → 11.6).

F. Stream-Factorized Attention Layouts

To make the attention-cost analysis in §3 and the ablation in Table 4 reproducible, Table 12 lists the per-model layer composition used for Stream-Factorized Attention. Intra-stream layers attend only within a stream. Cross-stream layers mix information across streams and follow the base model's attention pattern: "local" denotes sliding-window cross-stream attention, and "full" denotes full cross-stream attention.

The three layer types differ sharply in cost over the expanded length-nL sequence: an intra-stream layer costs $O(nL^2)$, a local cross-stream layer costs $O(nLw)$ for window size w, and a full cross-stream layer costs $O(n^2L^2)$. Full cross-stream layers are the only layers with quadratic growth in n, so the layouts keep their number small while still letting every token mix across streams. For base models that already interleave sliding-window and full attention (21B and

Table 11 | **Additional benchmarks for WeLM-HD4-80B.** The highlighted column is WeLM-HD4-80B, compared against the matched non-HD WeLM-80B under the same early SFT-only post-training recipe as Table 2. †External evaluations outside our harness.

Benchmark	WeLM 80B	WeLM-HD4 80B
Terminal-Bench 2	44.9	58.4
ARC-AGI-2	6.9	11.6
τ^2-Bench†	75.4	78.9
GDPval†	39.6	42.9
PolyMath	59.4	62.3
IFBench	70.7	73.0
LiveCodeBench v6	83.7	85.1
CritPt	3.7	4.9
AA-OmniScience	27.7	27.6
AA-LCR	69.3	69.3

Table 12 | **Per-model Stream-Factorized Attention layouts.** "Local" denotes sliding-window cross-stream layers (used when the base model has SWA); "Full" denotes full-attention cross-stream layers. The three 21B rows correspond to the configurations ablated in Table 4.

Model	Layers	Intra	Local	Full
21B, all-full	27	0	0	27
21B, SF (4 full)	27	10	13	4
21B, SF (1 full)	27	13	13	1
80B	49	20	23	6
617B	94	69	0	25

80B), the local cross-stream layers reuse the base model's sliding-window layers; the 617B base uses full attention throughout, so its cross-stream layers are all full (25 of 94), with the rest converted to intra-stream.

G. Progressive Expansion: Full Results

To show the full expansion-factor trajectory behind §4.3, Tables 13 and 14 report results on the 80B MoE that yields WeLM-HD4-80B at n=4 and on the dense Qwen3-8B-Base. Only the embedding tables grow with n; the backbone and the active computation per token are held fixed. The best value in each row is in bold.

We introduce the higher factors during continued pretraining at scheduled token counts, letting each factor converge before the next. On the 80B MoE, we expand to n=2, 4, and 8 after 0, 503B, and 906B tokens of the 32k continuation phase. The tables report two complementary signals: a steady reduction in language-modeling loss (Pile-test BPB on the 80B model, from 0.386 to 0.378) and consistent downstream gains, with n=8 best on almost every benchmark. The few exceptions are non-monotonic only at intermediate factors (e.g., MBPP+ on 80B dips at n=2 before recovering), while the n=8 model remains the strongest overall.

Table 13 | **Progressive expansion on the 80B MoE toward WeLM-HD4-80B.** Active parameters per token are fixed at 3B; only embedding parameters grow with n. BPB: lower is better.

Benchmark	**Shots**	**Base**	n=2	n=4	n=8
Embedding params	–	6.1B	12.1B	24.2B	48.4B
Training tokens	–	1.37T	503B	906B	1.01T
Pile-test (BPB) ↓	–	0.386	0.387	0.382	**0.378**
BBH (EM)	3	87.5	88.3	90.0	**90.6**
MMLU (EM)	5	85.1	85.0	86.7	**87.5**
C-Eval (EM)	5	88.8	88.9	**89.9**	89.5
SimpleQA	5	16.7	15.1	17.2	**18.4**
Chinese SimpleQA	5	60.7	62.2	63.8	**64.7**
HumanEval+	1	61.0	61.0	59.1	**62.2**
MBPP+	1	67.7	64.4	70.2	**71.2**
MATH	4	60.4	58.4	71.4	**71.7**

Table 14 | **Progressive expansion on the dense Qwen3-8B-Base.** Total parameters are fixed at 8B; only embedding parameters grow with n.

Benchmark	**Shots**	**Base**	n=2	n=4	n=8
Embedding params	–	1.2B	1.9B	3.1B	5.6B
Training tokens	–	180B	75B	150B	187B
BBH (EM)	3	78.8	81.3	83.0	**83.9**
MMLU (EM)	5	79.8	80.9	81.9	**82.2**
ARC-C	25	93.9	94.3	94.4	**94.7**
HellaSwag	10	79.7	83.1	85.0	**85.3**
GSM8K	4	92.5	93.3	93.9	**94.6**
MATH	4	56.0	58.2	60.0	**61.1**
MBPP+	1	66.7	69.4	68.7	**69.4**

H. CPT Startup Loss under Attention Compositions

To test whether using more intra-stream layers gives a smoother starting point for continued pre-training, we compare three n=4 attention compositions on a separate 617B-scale checkpoint. All runs start from the same step-16000 checkpoint before HD expansion and enable the expanded streams at the next step. The model, data, training schedule, and expansion factor are matched; the variants differ only in attention composition: no intra-stream layers, a 1:1 cross-/intra-stream layout, and a 1:3 cross-/intra-stream layout.

Figure 9 shows the loss immediately after the expansion is enabled. At the first expanded step, the losses are 1.59 for no intra-stream layers, 1.45 for the 1:1 layout, and 1.14 for the 1:3 layout. The layout with no intra-stream layers is higher than the 1:3 layout by +0.45 at step 1 and still by +0.33 at step 6. The ordering no intra-stream > 1:1 > 1:3 holds throughout the shared six-step window.

The run without intra-stream layers and the 1:1 run stop after 6 and 8 steps, so this check is

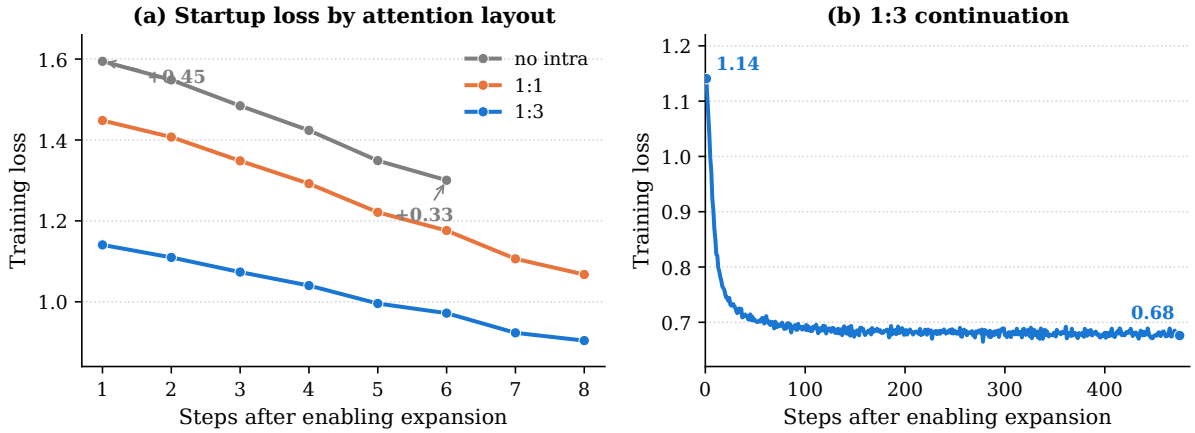

Figure 9 | **CPT startup loss under different attention compositions.** All runs enable n=4 expansion from the same 617B-scale step-16000 checkpoint. **(a)** The startup window compares no intra-stream layers, a 1:1 cross-/intra-stream layout, and a 1:3 cross-/intra-stream layout. The run without intra-stream layers and the 1:1 run stop after 6 and 8 steps, so the comparison is restricted to early startup behavior. **(b)** The 1:3 layout continues training for 475 steps.

limited to startup behavior rather than final convergence. The 1:3 layout continues training for 475 steps. These results support the design role of intra-stream layers: they reduce attention cost and also reduce the startup loss increase when the expanded model starts CPT from a checkpoint before HD expansion.

I. Base-Model Results with and without Hidden Decoding

To check whether the gains appear before post-training, Table 15 compares each non-HD Base model against its n=4 Hidden Decoding Base model.

Table 15 | **WeLM Base models with and without Hidden Decoding.**

	80B MoE		617B MoE	
Benchmark	**WeLM 80B-Base**	**WeLM-HD4 80B-Base**	**WeLM 617B-Base**	**WeLM-HD4 617B-Base**
MMLU	88.0	88.9	89.85	90.22
MMLU-Pro	68.4	71.3	72.26	73.92
C-Eval	91.3	91.4	91.9	92.5
SuperGPQA	50.5	52.6	55.33	57.05
BBH	90.7	91.9	92.41	93.07
MATH	63.6	62.6	70.92	70.82
SimpleQA	14.9	15.3	41.47	41.75
Chinese SimpleQA	56.0	57.4	78.13	78.8
AA-OmniScience	22.2	24.9	36.78	37.12
HumanEval+	70.1	72.0	76.2	76.2
MBPP+	70.9	71.4	71.4	70.9
Average	62.42	**63.61**	70.60	**71.12**

Hidden Decoding improves both scales before post-training. The average rises from 62.42 to 63.61 on 80B and from 70.60 to 71.12 on 617B. The larger gains appear on harder reasoning and knowledge tasks such as SuperGPQA and MMLU-Pro, and Table 2 shows that they persist after the same lightweight post-training recipe.