

Frontier Pre-training at Microsoft AI

In June, we announced MAI-Thinking-1, a general-purpose foundation model post-trained on top of MAI-Base-1 with strong reasoning performance. We’ve recently completed pre-training our new state-of-the-art model, MAI-Base-2, an MoE foundation model with 80 billion active and 2.8 trillion total parameters.

MAI-Base-2 outperforms every open-weight model we tested on our held-out evaluations. It is 4-12x more pre-training FLOPs-efficient than the strongest open base model, DeepSeek-V4-Pro-Base (see Figure 3 below), and we have put that saving into training a larger model.

Figure 1 shows the performance of MAI-Base-2 and other models with at least one trillion total parameters across a range of pre-training evaluations (lower is better).

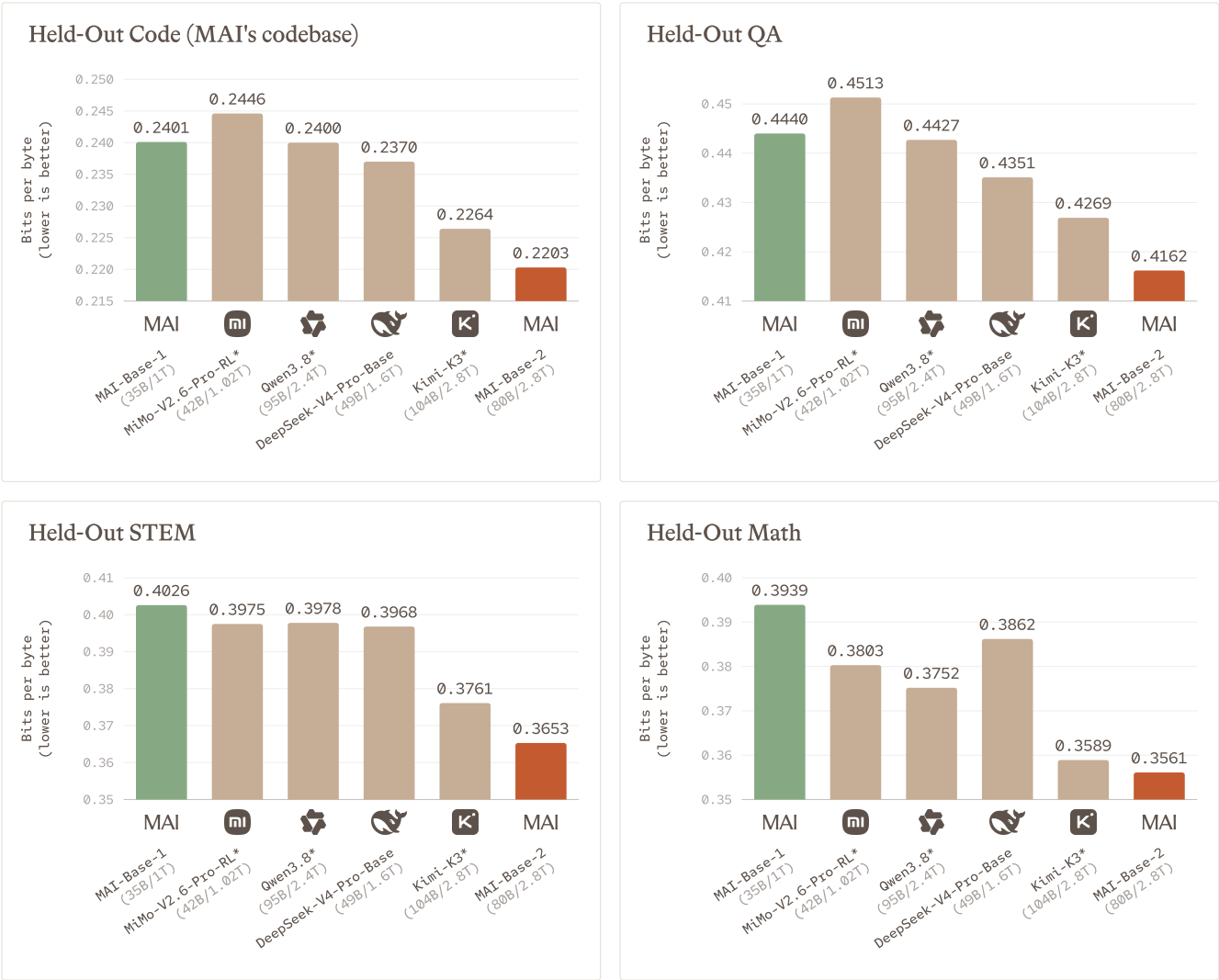


Figure 1: Held-out bits per byte (lower is better) on internal code, QA, Math, and STEM evaluation sets. *MiMo-V2.6-Pro-RL, Qwen3.8-2.4T-A95B, and Kimi K3 are post-trained models.

MAI

Owning our model pipeline end-to-end allows Microsoft to make a distinctive offering to our customers. Microsoft has strong data provenance over training tokens that go into our model. Our pre-training is built on organic data which does not involve distillation from other models. This will enable us to license our model weights to trusted partners through Foundry, which provides enterprises with a trusted source of world-class intelligence that serves their needs.

Owning the full stack allows us to tune our models and infrastructure together for the best performance at the lowest cost, first in our own products and then for Foundry customers. MAI-Base-2 is currently undergoing post-training, and we look forward to sharing more results soon.

How MAI-Base-2 Compares

When we finished training MAI-Base-1, our path ahead for pre-training was clear: scale up. While MAI-Base-1 is a strong model for its size, it is smaller than open weight and proprietary models at the frontier.

Three initiatives were particularly important for the successful pre-training of MAI-Base-2:

- Scaling science: scaling ladders that account for repeated training data, alongside empirical hyperparameter scaling laws.
- Model-infrastructure co-design: improved MoE and Muon implementations, alongside activation gradient clipping.
- Scaling data: expanded data pipelines and the integration of vision data.

MAI-Base-2 is a larger, sparser model than MAI-Base-1, trained on more data. We trained MAI-Base-2 on 16k GB200 GPUs over 38 days in August and September 2026. Over the course of the run, the average MFU (model FLOPs utilization) was 21% and the goodput (the fraction of time training progressed) was 89%. Our MFU is normalized by 2.5 PFLOPS (GB200’s theoretical max in BF16) and excludes FLOPs for rematerialization and the optimizer.

	MAI-Base-1	MAI-Base-2
Active / total parameters	35B / 1T	80B / 2.8T
Layers	78	102
Hidden dimension	6656	8704
MoE layout	Alternating dense FFNs and MoEs	MoE in every layer
Dense FFN dimension	13312	n/a
MoE down projection dimension	3072	4096
MoE expert FFN dimension	10240	8704

	MAI-Base-1	MAI-Base-2
MoE routing / expert count	0 shared + 8 / 512 routed	1 shared + 7 / 512 routed (shared expert is not down/up projected)
Attention heads (KV / Q)	- 8 / 80 (global - every 6 layers) - 8 / 80 (local)	- 2 / 96 (global - every 6 layers) - 16 / 96 (local)
Embeddings	Tied	Untied
Training horizon (tokens)	30T	50T
Optimizer	AdamW	Muon (non-embedding 2D parameters), AdamW
FP8 scheme	Tensor-wise, delayed scaling E4M3 / E5M2	MXFP8 E4M3
Gradient clipping	Global weight gradient clipping	Global weight gradient + activation gradient clipping
Vision	Text only	Native VLM

We compare pre-trained models on held-out text using bits-per-byte (BPB), which is agnostic to each model’s tokenizer. For held-out code, we use MAI’s own codebase, while STEM, Math, and Q&A evaluations are commissioned from vendors to reduce the risk of training-set contamination. We additionally decontaminated all evaluation sets against our training data.

We compare MAI-Base-2 with frontier checkpoints available before October 2026:

- DeepSeek-V4-Pro-Base is the most capable base model with open weights.
- Kimi K3, Qwen3.8-2.4T-A95B, and MiMo-V2.6-Pro-RL are highly capable open weights models. Only post-trained checkpoints are available for these models. We report their results with the caveat that post-training may affect these evaluations.

Across these held-out evaluations, MAI-Base-2 achieves lower BPB than the other models we tested. Below, we dive into a few of the advances that made this possible.

Robust scaling ladders

We evaluate model changes using a scaling ladder, a family of models of different sizes with a fixed training recipe. Each rung of the ladder is named after its number of layers L , and L alone determines the rest of the architecture. For example, the hidden size is $(256/3)*L$, and the number of query heads is L , rounded to the nearest multiple of 16. We

MAI

train models of increasing size while holding training tokens per active parameter (TPP) constant.

To account for multi-epoch training at smaller scales, we use a technique we call *source scaling*. On the lower rungs of the scaling ladder, source scaling artificially restricts the set of unique samples the model trains on, so that each dataset is repeated the same number of times regardless of the training horizon. For example, if a dataset is repeated 8 times over the full 50T-token run, a 5T-token run will loop 8 times over just 1/10 of that data.

Under a fixed TPP, source scaling effectively keeps the ratio of unique data to parameters fixed, which helps ensure that results transfer smoothly across scales. Without source scaling, experiments at smaller scale can favor under-regularized architecture and hyperparameter choices that overfit at larger scales. Including source scaling in our ladders was key to scaling up to a 2.8T parameter model reliably. Figure 2 demonstrates the use of a scaling ladder to accurately forecast the performance of MAI-Base-2.

MAI-Base-2 Scaling Ladder

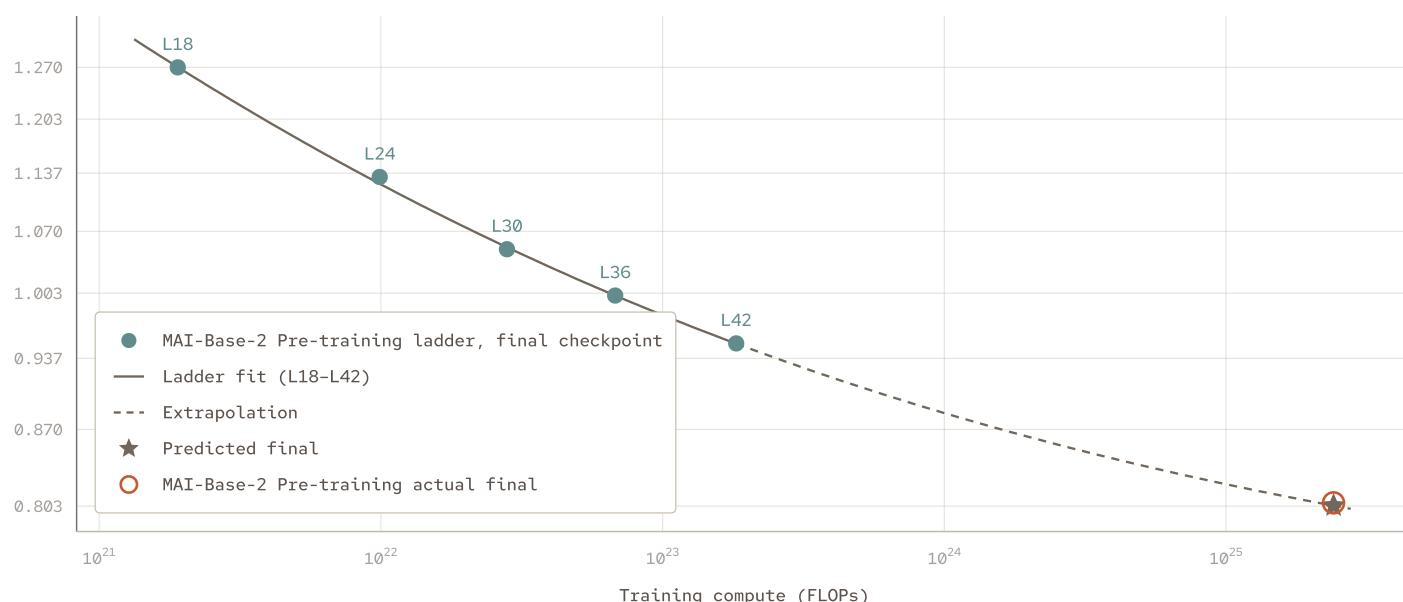


Figure 2: Our scaling ladder accurately forecast MAI-Base-2’s performance.

Isolating modeling efficiency from compute

Direct comparisons between models conflate the quality of the training recipe with differences in model size and training compute. To isolate the recipe, we estimate how much compute our recipe would need to match a strong external base model. We use DeepSeek-V4-Pro-Base as the reference: it is the only released base pretraining checkpoint with more than one trillion parameters, though it is still trained on significantly fewer FLOPs than MAI-Base-2.

For each of the four held-out evaluations in Figure 1, we fit an independent scaling law of BPB against training FLOPs and estimate the compute a model on our ladder would need to match DeepSeek-V4-Pro-Base’s BPB. MAI-Base-2’s final performance is known, and we include it in the fits, so the matching point falls within the range of our fitted models rather than beyond it. Comparing these estimates with DeepSeek-V4-Pro-Base’s training compute, we find that

our recipe is 4.2–12.3x more FLOPs-efficient, as shown in Figure 3. We also compare against MAI-Base-1 and find that our progress since that model’s release in June this year has led to 6.3-12.7x FLOPs-efficiency improvement. We attribute MAI-Base-2’s FLOPs-efficiency to the quality and relevance of our data and to the efficiency gains from our modeling choices, both described in the sections below.

BPB Scaling: MAI Ladder vs DS-V4-Pro and MAI-Base-1

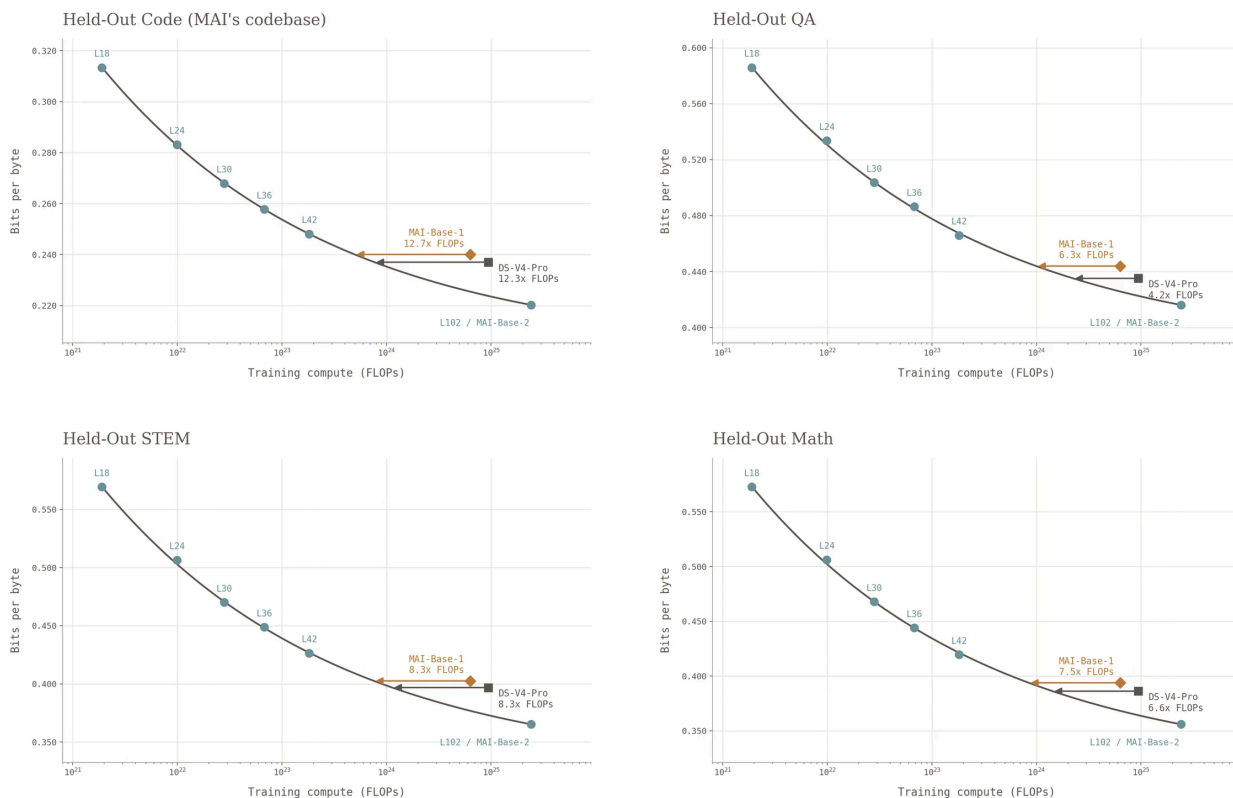


Figure 3: Projections of DeepSeek-V4-Pro-Base and MAI-Base-1 onto MAI-Base-2’s scaling law across held-out evaluations. MAI-Base-2 model family is able to achieve the same bits per byte as these two baselines with significantly less training compute.

Scaling laws for hyperparameters

Selecting optimal hyperparameters for a large training run is complicated by the combinatorial space of hyperparameter optimization, where changing one hyperparameter can affect the optimal value of others. For MAI-Base-2, we revisited how we approach this problem.

We swept learning rate (LR) and weight decay (WD) jointly because their effects on training are coupled. Model performance is most sensitive to the product of LR and WD , so good configurations lie in a narrow diagonal band of the $(\log LR, \log WD)$ plane. A rectangular grid in $(\log LR, \log WD)$ would sample this band inefficiently, so we instead swept a grid in $(\log LR, \log (LR \times WD))$.

For several combinations of model size and TPP, we trained models along the search grid and evaluated them on our pre-training evaluations. We fit a 2D quadratic function to evaluations and extracted the analytical minimum ($LR^*, LR^* \times WD^*$). Figure 4 shows a few slices of this experiment.

Weighted eval NLL – quadratic surface fit

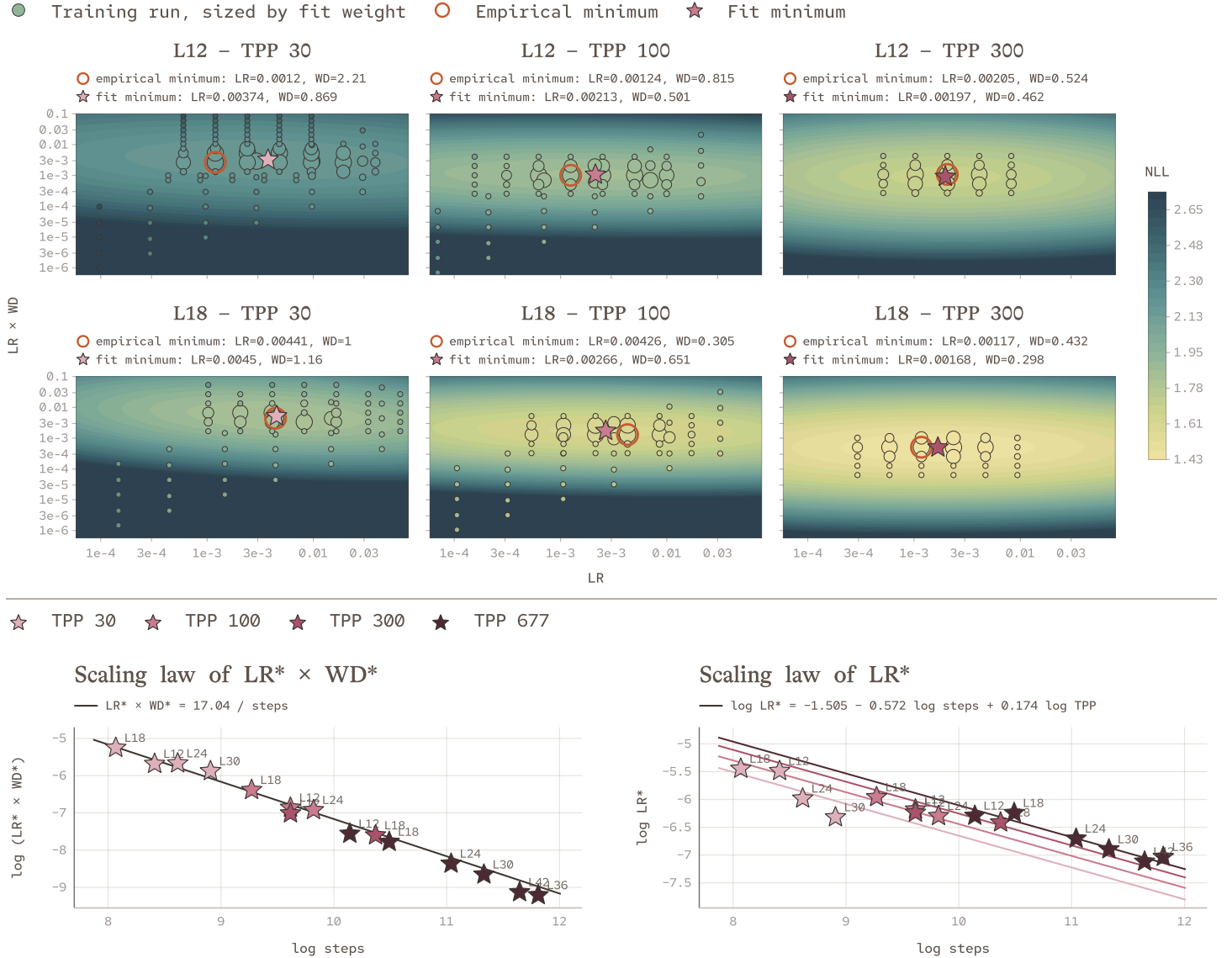


Figure 4: Top rows: $(\log LR, \log(LR \times WD))$ sweeps for multiple model sizes and tokens per parameter. Each circle indicates a training run, and its color shows its measured negative-log-likelihood (NLL). The size of the circle shows the weight of the data point when fitting a 2D quadratic, whose prediction is shown as the background on the same color scale. The weight decreases with the relative gap between the run’s NLL and the minimum NLL in that slice. Bottom row: scaling laws for $LR^* \times WD^*$ and LR^* .

We then modeled $\log(LR^* \times WD^*)$ and $\log LR^*$ as a function of model size, TPP, or other features such as log steps. Using model selection, we found that $LR^* \times WD^*$ depended primarily on the number of training steps, while $\log LR^*$ additionally depended on $\log TPP$. The empirical exponent for $LR^* \times WD^*$ with respect to steps was very close to -1 , which matches the view that the decoupled weight decay time scale ($\frac{1}{LR \times WD}$) should be a fixed

MAI

fraction of training. The empirical exponent of LR with respect to steps was close to -0.5 , which matches the widely used rule of thumb that when doubling the batch size, one should multiply the learning rate by $\sqrt{2}$. Our final scaling laws for LR^* and WD^* are:

$$LR^* \times WD^* = 17.04/\text{steps}$$
$$\log LR^* = -1.51 + 0.175 \log \text{TPP} - 0.573 \log \text{steps}$$

Using a similar approach, we also derived the following scaling law for dropout:

$$\text{Dropout} = \text{sigmoid}(0.526(\log \text{ActiveParams} - 27.2))$$

We estimated our new scaling laws in small-scale experiments, then validated them at an intermediate scale where they consistently outperformed previous heuristics. These laws defined MAI-Base-2's hyperparameters.

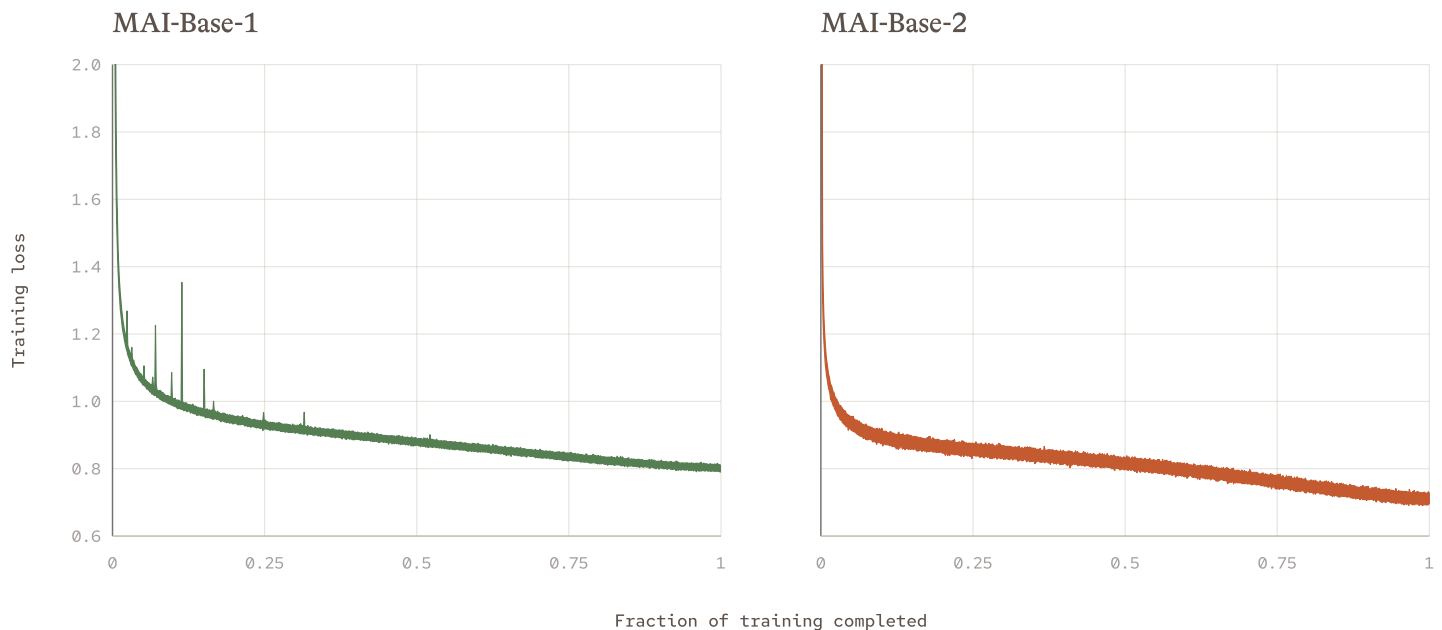


Figure 5: Training loss for MAI-Base-1 and MAI-Base-2. MAI-Base-2’s training was smooth over the full training horizon. We credit that to our focus on careful scaling ladders and incremental refinement of our training recipe.

Codesigning our infrastructure and MAI-Base-2

Training frontier models within a compute budget requires co-designing models and infrastructure for both evaluation quality and computational efficiency. We use an efficiency gain (EG) metric to help guide our decisions: the ratio of compute required by a candidate to reach the same evaluation loss as the baseline. Compute can be measured in FLOPs (EG_{FLOPs}) or in wall-clock time (EG_{time}), and the two are complementary. We rely mainly on EG_{time} , since it

reflects the current state of our infrastructure. EG_{FLOPs} , in turn, measures a prototype's potential and guides future infrastructure development: a change that wins on FLOPs but loses on wall-clock time shows us where to invest. This joint optimization of modeling and infrastructure is reflected in several design choices below.

Scaling MoE

Our past modeling experiments have shown that MoE in every layer has a consistent EG_{FLOPs} advantage over alternating dense FFN and MoE layers. However, inefficiencies in our MoE implementation made MoE every other layer EG_{time} optimal for MAI-Base-1. For MAI-Base-2, we identified two inefficiencies that led to suboptimal MoE MFU:

1. Dynamic routing increased memory fragmentation and tied up more memory for rare, extreme expert load imbalances, memory that could otherwise reduce activation recomputation.
2. Adding more, finer-grained experts increased per-expert overhead.

Addressing these inefficiencies increased the MFU of our MoE layers from 24% to 35%, which allowed us to adopt MoE in every layer for MAI-Base-2.

Droptless MoE with tunable, fixed-size memory usage

In MoE layers, each expert sees a variable number of tokens. This introduces GPU memory challenges as experts that are overloaded can run out of memory. One solution is to set a fixed threshold for each expert, after which additional tokens are dropped. Our experiments found that dropping tokens leads to worse performance. A naive solution is to simply allocate enough memory per expert to account for the largest imbalance, but this leads to inefficient usage of memory under normal loads. Our multi-round droptless MoE, illustrated in Figure 6, achieves fixed memory usage under arbitrary routing imbalance, eliminating fragmentation and out-of-memory crashes.

EG = expert group • D = dispatch (a2av) • C = collect (a2av)

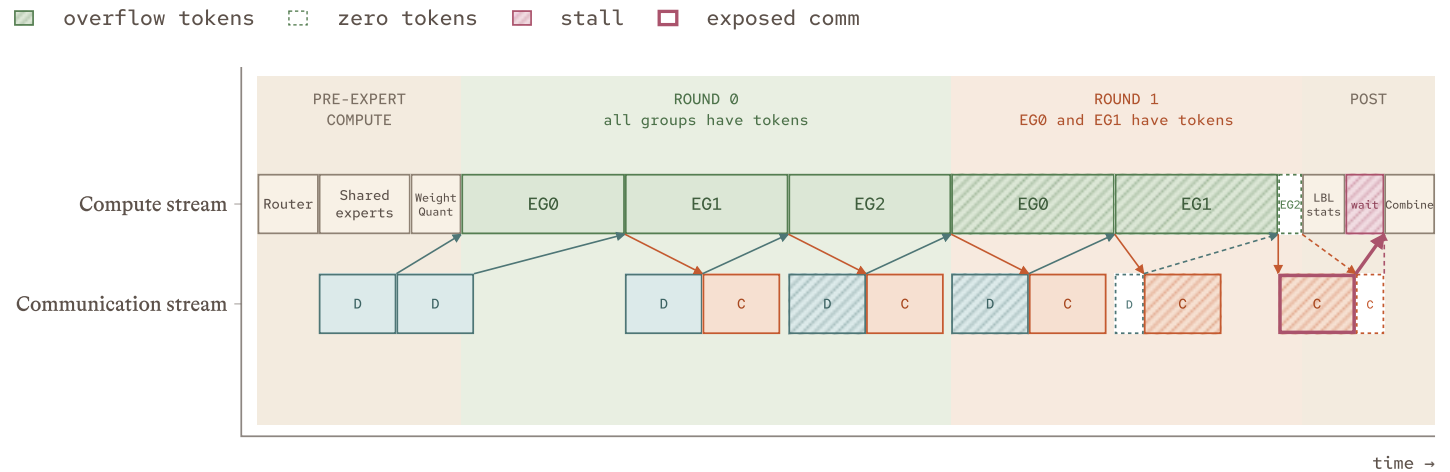


Figure 6: Forward pass of multi-round droptless MoE with three expert groups, where expert groups 0 and 1 (EG0 and EG1) overflow. The round count is global, so EG2 runs during the second round despite having zero-tokens (compute is skipped). MFU drops during extra rounds, but this is acceptable because steady-state only has one round.

MAI

We group the experts on each expert parallel rank into one or more groups of a configurable size. Each expert group runs one or more *dispatch* $\rightarrow$ *compute* $\rightarrow$ *collect* rounds with a fixed per-expert token capacity. If an expert receives more tokens than it can hold, the expert group processes those overflow tokens in additional rounds. To reduce the number of additional rounds, we allow overloaded experts to use unused token capacity from other experts in the same group. In the backward pass, we then perform fine-grained recomputation per expert group following a similar multi-round schedule. This keeps memory usage static regardless of token imbalance. Token capacity is configurable: more capacity means more memory, and less capacity means potentially more rounds and thus extra compute. In practice, the capacity is configured to maintain a single round in steady state.

Dispatch, *compute*, and *collect* are pipelined across all rounds of all expert groups, so that all but the first *dispatch* and the last *collect* overlap with expert compute. In the forward pass, weight quantization and shared-expert compute overlap with the exposed communication. In the backward pass, all expert communications are fully overlapped: the exposed input *dispatch* is overlapped with recompute of the same layer, and the exposed gradient *collect* is overlapped with weight-gradient computations.

Reducing per-expert overhead

We reduce per-expert overhead in four ways. First, we improve GEMM and NVLink efficiency by grouping both computation and communication for each expert group. This reduces kernel-launch overhead and tail effects. Second, we reduce host-device synchronization by using device-resident token metadata and initiating all-to-all communication directly from the GPU. Third, we further optimize all-to-all with symmetric memory to exchange tokens directly over NVLink without staging copies, improving communication throughput and reducing HBM contention when running GEMMs concurrently. Lastly, we batch load-balancing loss across layers, replacing per-layer all-reduces with a single batched reduction across all layers. These optimizations enable us to train sparser models without significantly degrading MFU.

Scaling Muon

When scaling up to MAI-Base-2, we found that Muon delivered more stable training dynamics and large efficiency gains over AdamW. Adopting Muon for our largest models required reducing its per-step overhead.

Muon orthogonalizes the momentum of each weight matrix with Newton–Schulz, an iterative algorithm consisting of several matrix multiplications. This requires the full matrix, which conflicts with ZeRO's sharding of momentum across the data-parallel dimension. Following prior work, we reshard the momentum during the optimizer step. Each GPU updates its local FP32 momentum shard and casts it to BF16. An all-to-all exchange, batched over parameters of the same shape, then gives each GPU ownership of a subset of full momentum matrices. Each GPU orthogonalizes its full momentum matrices locally with a symmetry-aware Newton–Schulz kernel. A second all-to-all returns the updates to their original shards, where they are cast back to FP32 and applied to the weights.

We further apply several optimizations that are beneficial to our training efficiency on 16K GPUs:

- We overlap communication and computation by prefetching multiple batches.
- We cache and reuse the resharding metadata to avoid repeated indexing logic. This is especially useful in sparse MoE models with thousands of identically shaped parameters.

MAI

- Selected weights, such as the MoE router weights, are replicated across GPUs rather than sharded. The first communication phase is skipped, and we instead issue an all-gather collective after Newton-Schulz to get the full set of replicated updates.

After optimizations, Muon accounted for 2.1% (165 ms) of the average training iteration time of MAI-Base-2. We tracked the quality of our implementation and prioritized changes based on profiles, using both large-scale and small-scale testing setups. We compared measurements from tests to the theoretical peak performance, which we estimated to be 55 ms and bottlenecked by communication for MAI-Base-2's final training setup. We attribute the gap between theoretical and measured performance to stragglers, kernel dispatch overheads, load imbalance, and memory copies/allocations during resharding.

Activation gradient clipping

Gradient clipping is a common method to stabilize training by reducing the impact of outlier gradients. Standard gradient clipping operates on the global norm of the weight gradient for a whole batch. However, outlier gradients are typically driven by a small portion of the data within a batch. An outlier can therefore cause the gradients from the entire batch to be scaled down, including contributions from otherwise well-behaved examples. As a more targeted mitigation, we clip the gradients of activations at selected places in the network. Together with Muon, this keeps MAI-Base-2's gradient norm in a much tighter range than MAI-Base-1's (Figure 7).

■ Per step — Rolling median (1,000 steps)

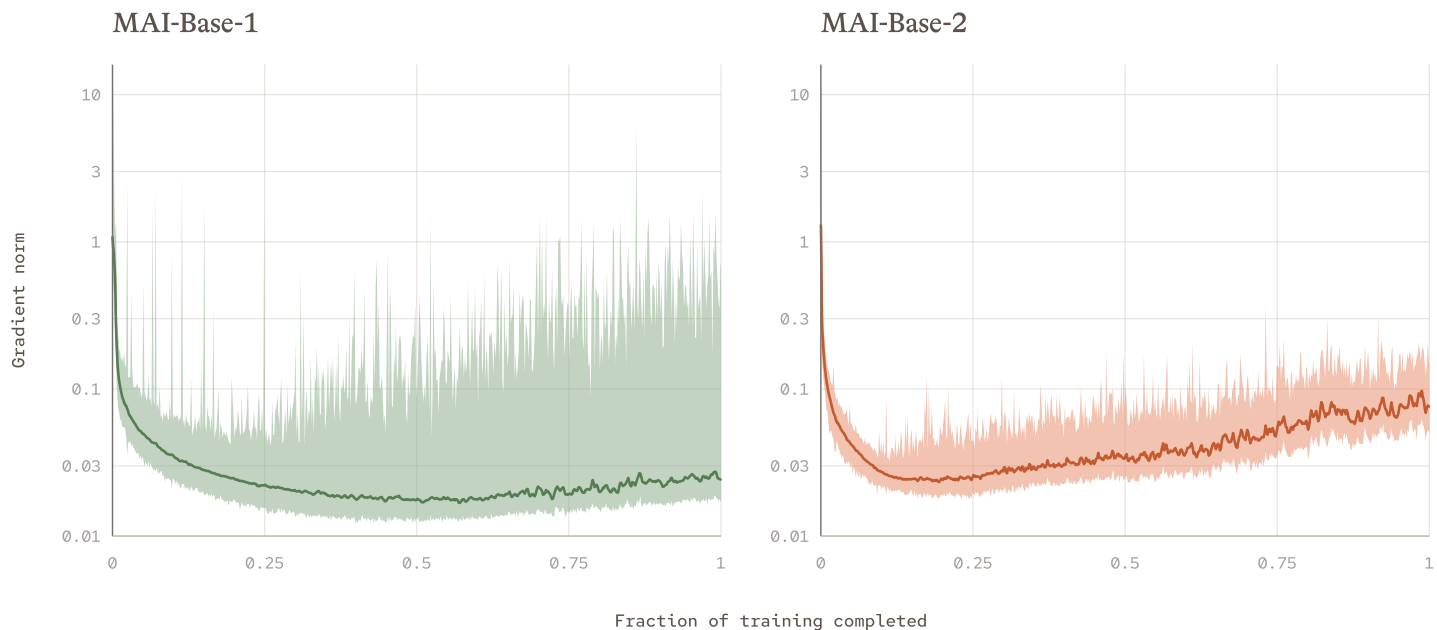


Figure 7: Training gradient norm for MAI-Base-1 and MAI-Base-2. MAI-Base-2's recipe changes, including activation gradient clipping and Muon, led to a much tighter range.

Activation gradient clipping also made optimization more stable, and its benefit grew with scale as shown in Figure 8. In contrast, most changes we have tested yield a roughly constant efficiency gain across scales.

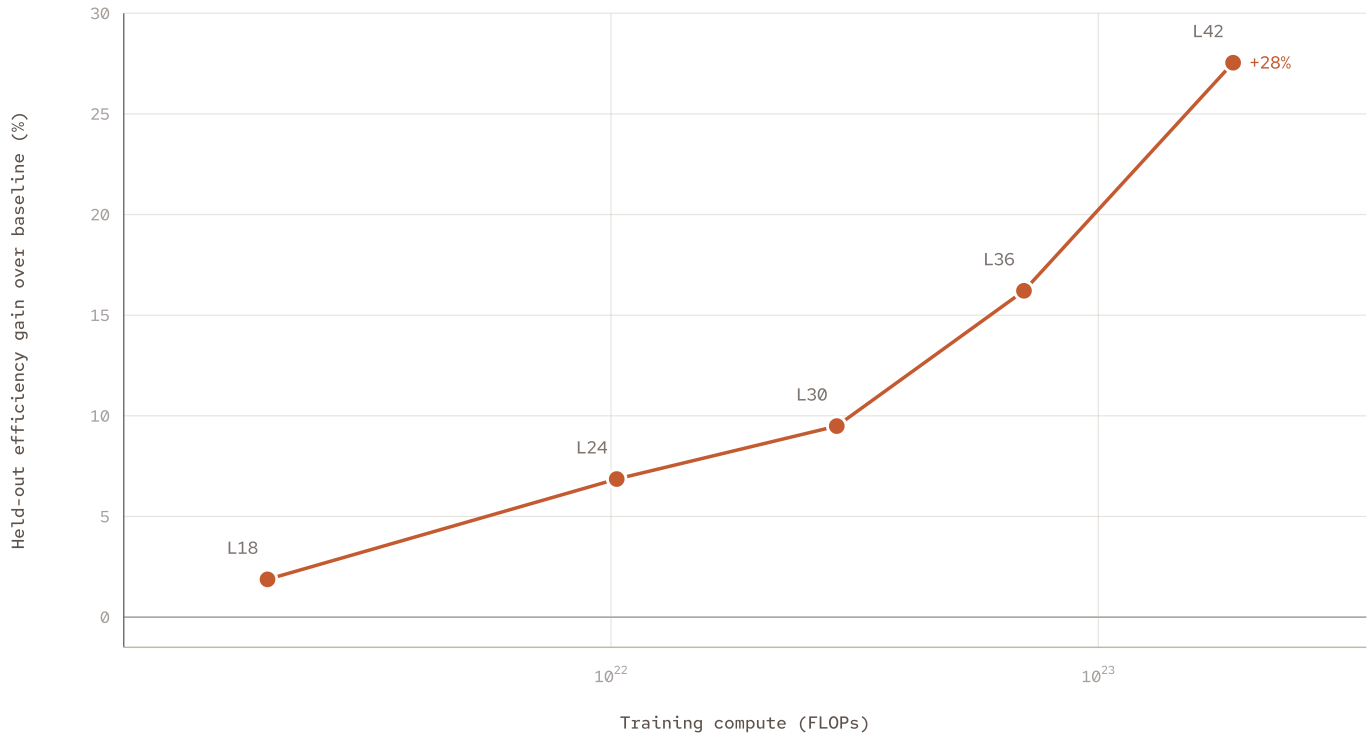


Figure 8: Efficiency gain (EG_{FLOPs}) on held-out evaluations from activation gradient clipping.

We also use activation gradient clipping as a diagnostic tool for validating our training data. Figure 9 shows a malformed training example identified by activation gradient clipping. In a pull request from our code data, a license header is abruptly cut off mid-sentence. Tracing this clipped example back revealed a bug in our preprocessing pipeline, which has since been fixed.

```
/*
 * Copyright 2010 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this copy file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under # PR Title
[pull] master from gradle:master

# PR Body
See
```

Figure 9: A training sequence from our code data flagged by activation gradient clipping. The highlighted “#” is the token whose activation gradient was clipped.

Scaling Data

MAI-Base-2 follows the same data principles as MAI-Base-1: our path to frontier performance is built on organic data, not distillation from other models. Investing in our own data pipelines continues to pay off, and we processed every training token in-house. Our corpus grew by roughly 36% from MAI-Base-1 to MAI-Base-2, measured in unique training tokens (see Figure 10).

Since MAI-Thinking-1, we've improved our quality classification, topic binning, web parsing, deduplication, and evaluation decontamination. For MAI-Base-2, we expanded our pipelines to process substantially more web content¹, with a March 2026 knowledge cutoff. We also used embedding-based diversity sampling to prioritize novel data and reduce duplicative content. Our experiments showed that these approaches substantially improved model performance.

Unlike MAI-Base-1, MAI-Base-2 is pre-trained natively with vision using an encoder-free VLM architecture. Accordingly, we invested heavily in growing our VLM corpus across image-text pairs and documents with interleaved text and images. The corpus spans tasks including image captioning from the web, chart understanding, visual grounding, spatial reasoning, medical understanding, computer use, sheet-music reading, and other areas we anticipate users will find useful. Our VLM corpus contains 4.5T unique tokens and, with repetition, VLM data accounted for 18% of the 50T-token training mixture.

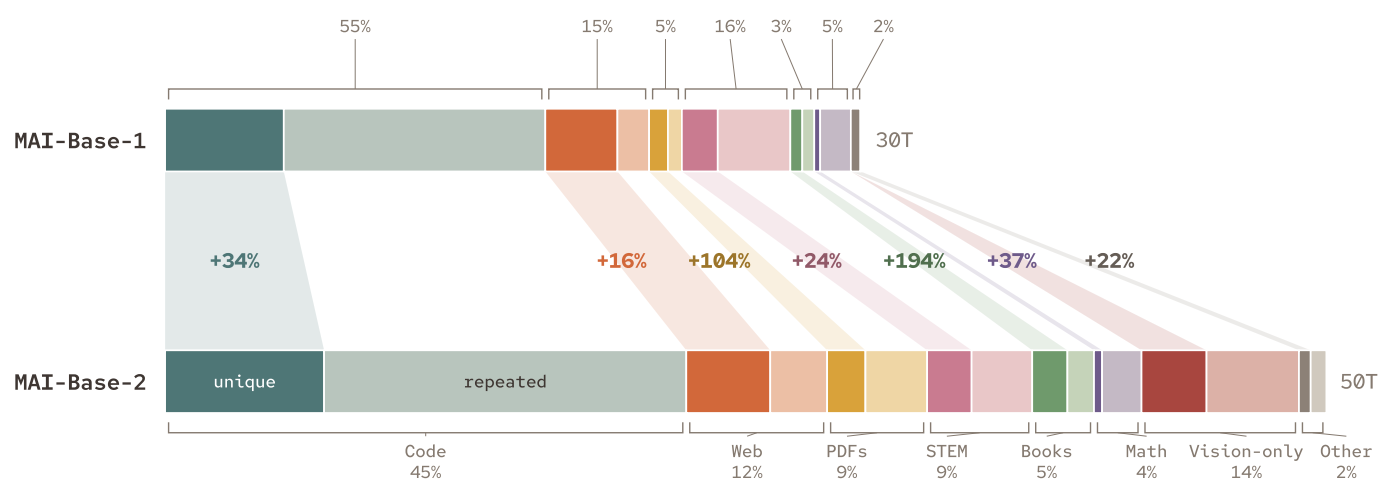


Figure 10: MAI-Base-2 pre-training mixture compared to MAI-Base-1. Solid: unique tokens, light: repeated tokens.

What's next

MAI-Base-2 is the foundation for our next generation of models, which we're excited to share more about soon. While MAI-Base-2 goes through post-training, preparations for MAI-Base-3 are already underway. All our models are built following the same principles that have guided us since MAI-Base-1:

- Intelligence should be learned from first principles.

MAI

- Simple methods scale best.
- Scientific rigor over shortcuts.

We have conviction that these principles will lead to continued scaling towards the frontier. Training frontier AI requires researchers and engineers across modeling, data, and infrastructure to work as one team. If this appeals to you, we're hiring!

-
1. For web data, we use a proprietary crawler that respects the Robots Exclusion Protocol (robots.txt) and related meta-tag and HTML controls, enabling site owners to manage how content on their sites is accessed and used. We also exclude sources that violate Microsoft Responsible AI policies or appear on the Office of the United States Trade Representative (USTR) Notorious Markets list.