Lecture 17

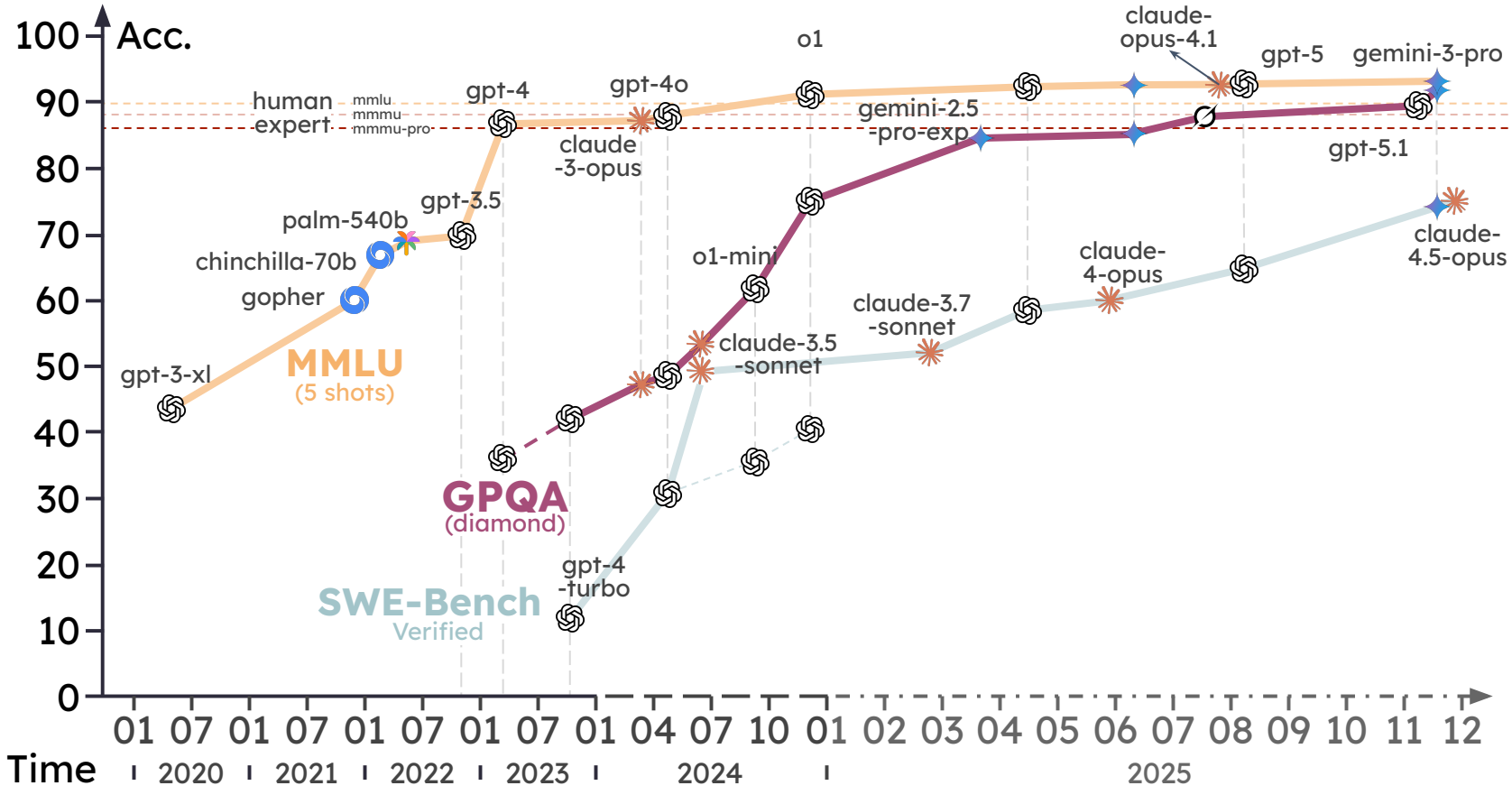
Open Recipes of Pre-training and Post-Training

https://cis7000-fall26.thelabone.org/

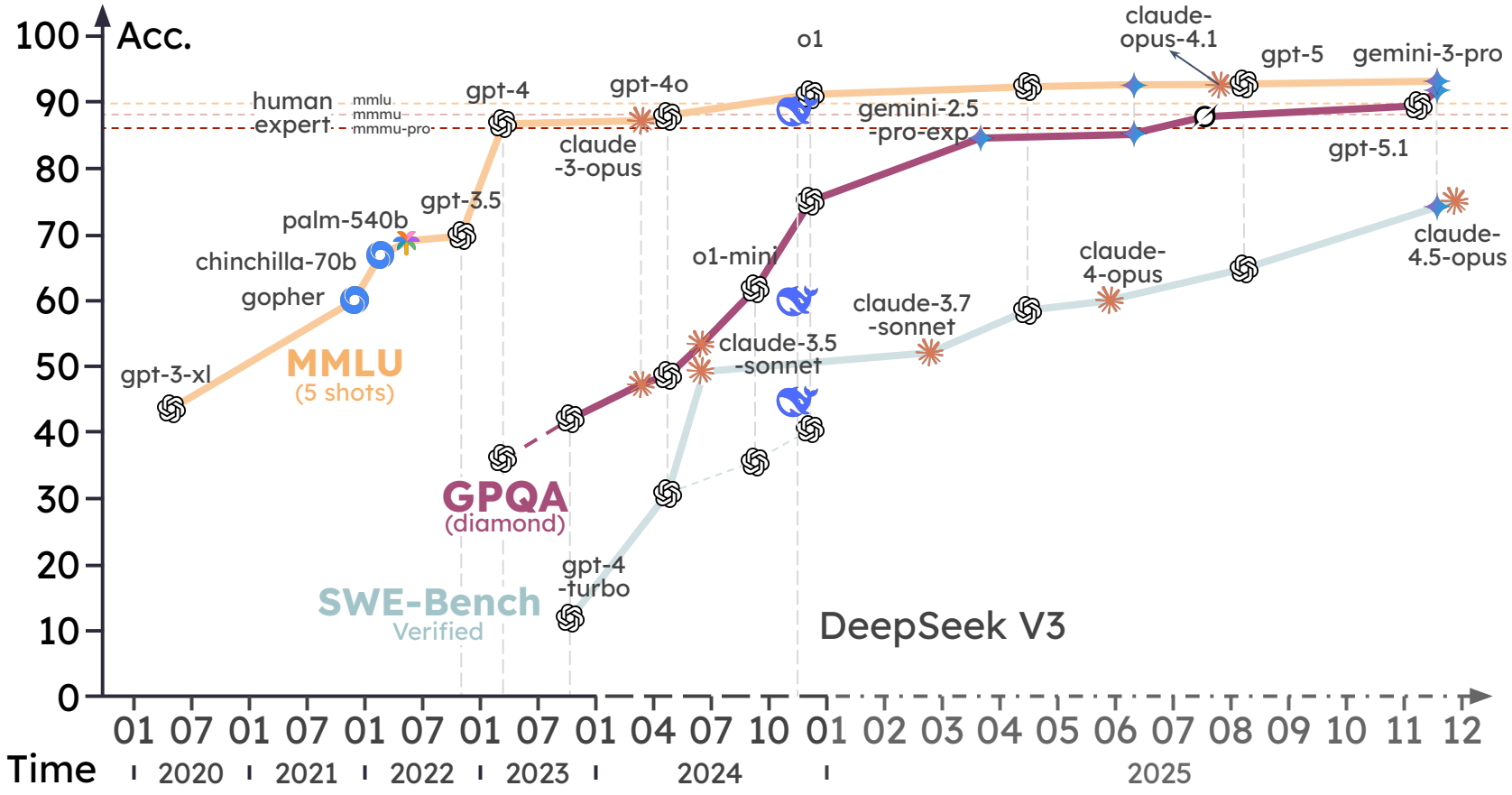
Martin Ziqiao Ma Oct 26th, 2026

(Views are my own, not of my employer)

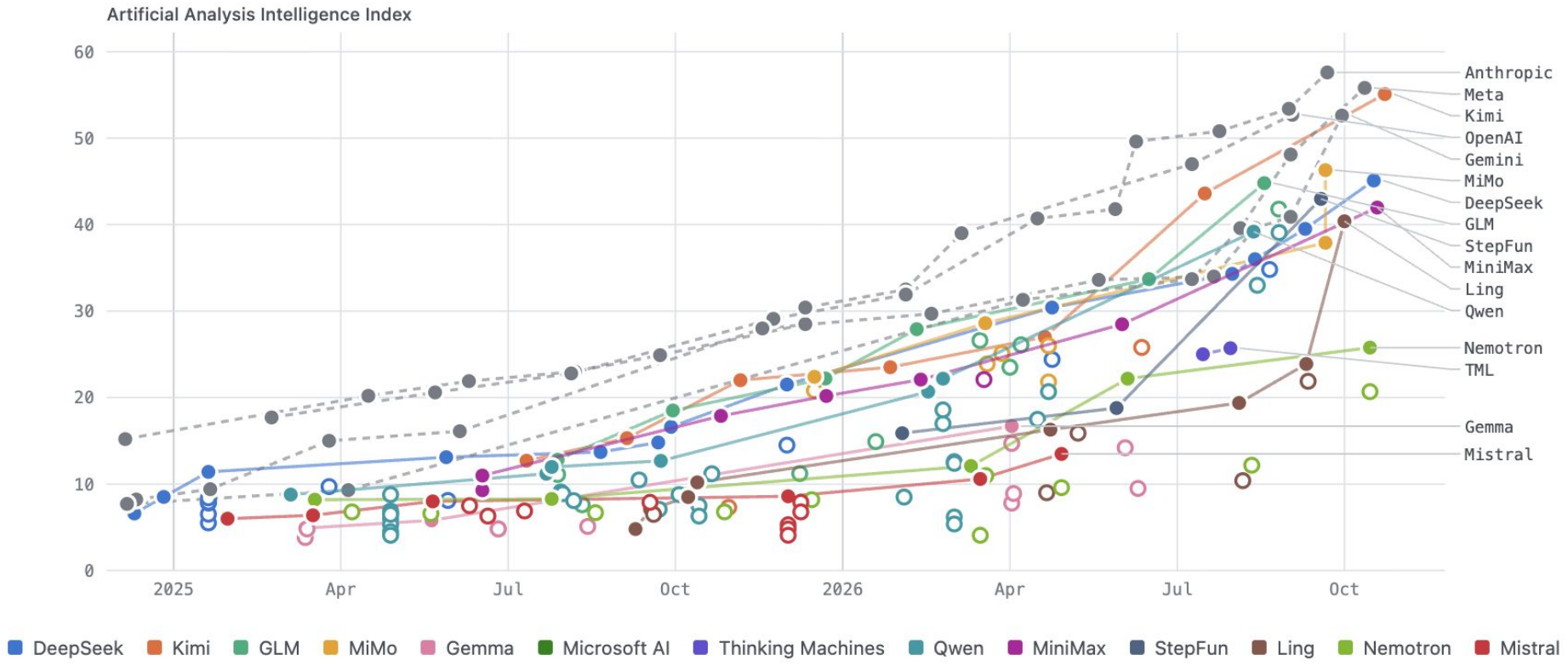
The Leaderboard Game



Open Models in the Leaderboard Game



Open Models in the Leaderboard Game



Why We Care About Open Recipes

- Open-weight technical reports have become the most detailed public record of how large models are trained, and released weights make the reported configurations checkable.
- Open models now trail the best closed models by a narrowing margin on common benchmarks.
- Labs build on each other's open models through fine-tuning, distillation and synthetic data, so recipes propagate across the field.
- Reports include evidence at scales few others can test, including negative results.
- This lecture: a summary of 13 labs, 170 releases and 414 public documents (technical reports, model cards, configuration files, blogs) between Dec 2024 – Oct 2026.

Terminology: Three Training Stages

- **Pretraining**: next-token prediction on a broad corpus, from random initialization, over one long learning-rate schedule
- **Midtraining**: further training of a pretrained model, usually with the same objective, but on targeted data or settings: curated and reasoning data, longer context, new modalities, architecture changes
- **Posttraining**: learning from demonstrations and rewards: SFT, preference optimization, RL, distillation
- **Boundary cases**: a phase counts as midtraining if the lab names it (annealing, context extension), if it is a separate stage after the main decay, or if it continues a finished base

Part 1: Pretraining



The Textbook Transformer Revised

Component	Textbook (2017–2023)	2026 open models
Feed-forward	Dense: every parameter for every token	Mixture-of-experts: about 3–10% of parameters active per token
Attention	Full multi-head attention in every layer	Hybrids of a few full or sparse layers with windowed, linear or Mamba layers; compressed KV caches
Context	2K–8K tokens	128K–1M tokens
Optimizer	AdamW	Muon at 8 of 13 labs
Precision	BF16	FP8 standard; 4-bit at NVIDIA

A Model Is Paid for Twice

- The training bill is paid once and grows with the parameters each token uses and the number of training tokens.
- The serving bill is paid for every generated token, and at long context it grows mostly with the memory read per token.
- The best pretraining architecture must also be economical at inference.

BILL 1

Train once

Tokens × active computation

Main resource:
accelerator compute (FLOPs).

BILL 2

Serve repeatedly

Stored weights + KV cache

Main resource:
memory capacity and traffic, as well as compute.

The Training Bill: 6ND

- Each weight takes part in three matrix products per token, one forward and two backward, at 2 FLOPs each.
- Training compute is $C \approx 6 \cdot N \cdot D$, where N is the active parameters and D the training tokens; serving costs about $2 \cdot N$ FLOPs per token.
  - DeepSeek-V3: $6 \times 37B \times 14.8T \approx 3.3 \times 10^{24}$ FLOPs, consistent with the 2.66M H800 GPU-hours its report gives.

Forward

2 FLOPs per active weight

Compute activations

Backward: weights

2 FLOPs per active weight

Compute weight gradients

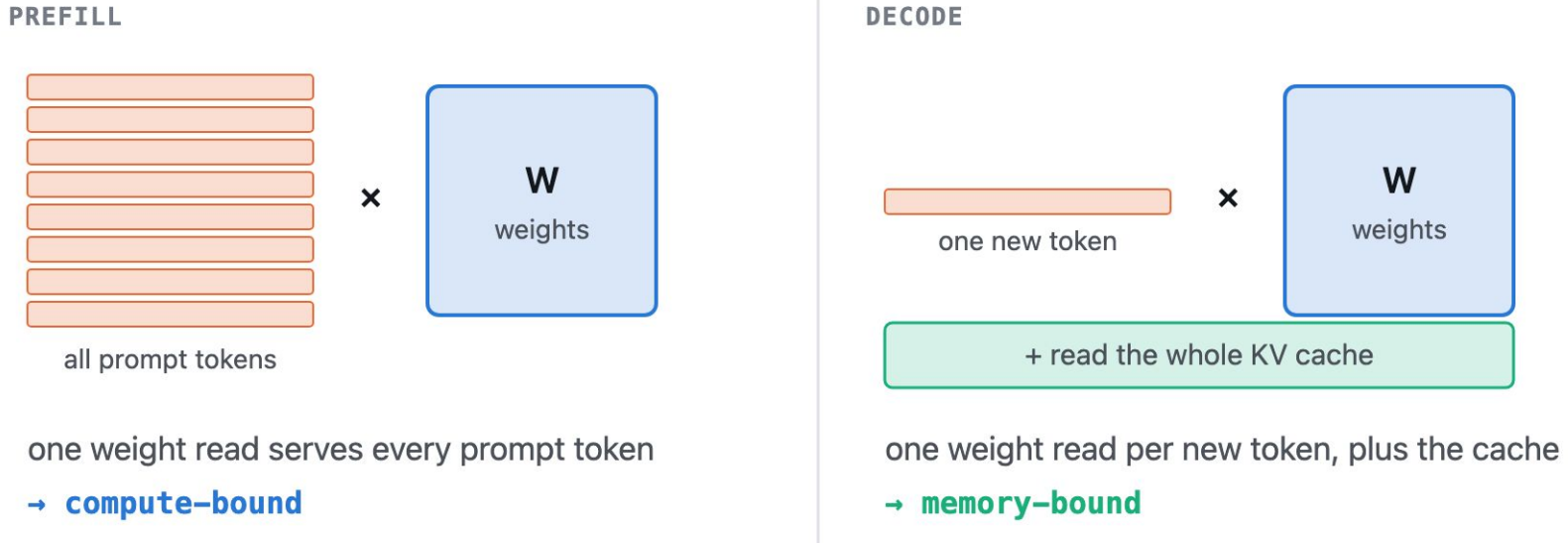
Backward: inputs

2 FLOPs per active weight

Pass gradients to earlier layers

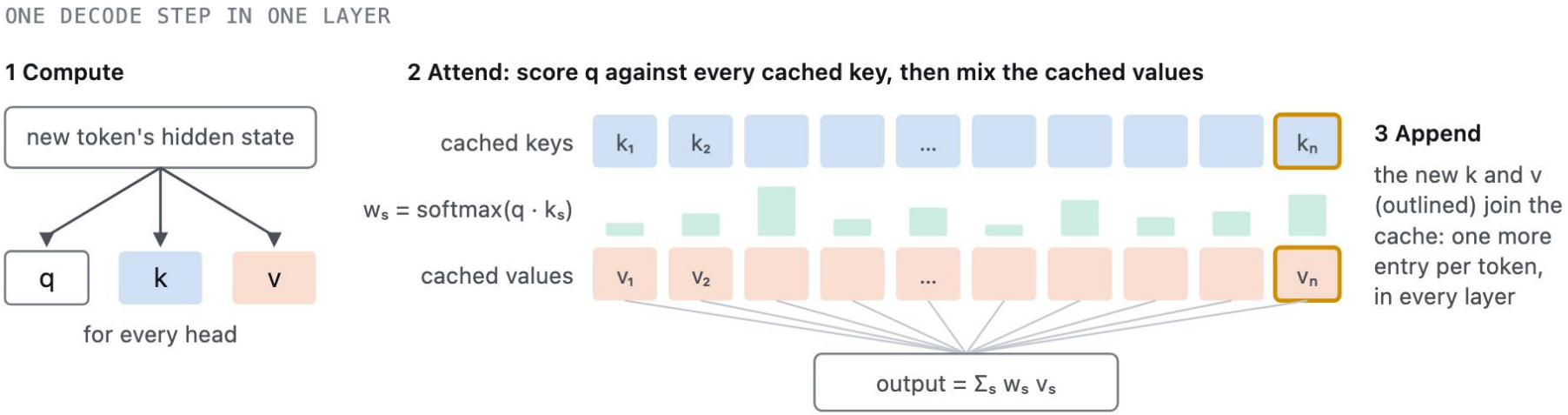
The Serving Bill: Prefill and Decode

- Prefill processes the whole prompt at once, so each weight read from memory is used for many tokens.
- Decode generates one token per step: each step needs the prev. step's output.
- Both spend about 2 · N FLOPs per token on weights; they differ in how many bytes each step must read.



The KV Cache

- Earlier tokens do not change, so their keys and values are stored instead of recomputed; this store is the KV cache.



Without the cache, step 2 would first recompute the key and value of every earlier token, in every layer, at every step.

The KV Cache

- Earlier tokens do not change, so their keys and values are stored instead of recomputed; this store is the KV cache.
- In every layer, one decode step:
  - computes the new token's query, key and value;
  - scores the query against every cached key and mixes the cached values;
  - appends the new key and value.
- The cache costs capacity, since it must fit beside the weights, and bandwidth, since it is reread at every step.

How Large the KV Cache Is

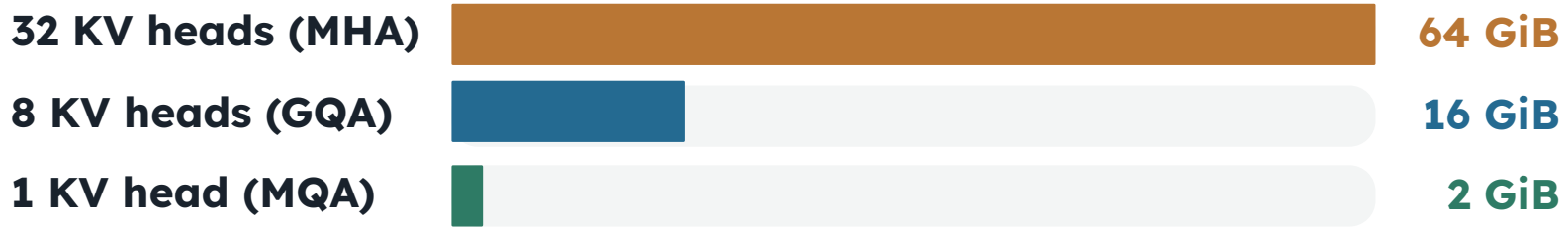
- KV bytes per token = 2 (key and value) × layers × KV heads × head dimension × bytes per number.

$$\text{KV bytes per token} = \underbrace{2}_{\text{key and value}} \times \underbrace{L}_{\text{layers}} \times \underbrace{h_{\text{kv}}}_{\text{KV heads}} \times \underbrace{d_{\text{h}}}_{\text{numbers per head}} \times \underbrace{b}_{\text{bytes per number}}$$

- Toy: 32 layers, 8 KV heads of dimension 128, bfloat16 → 128 KiB per token, 16 GiB for one 128K-token sequence.

Decode Is Limited by Memory, Not Arithmetic

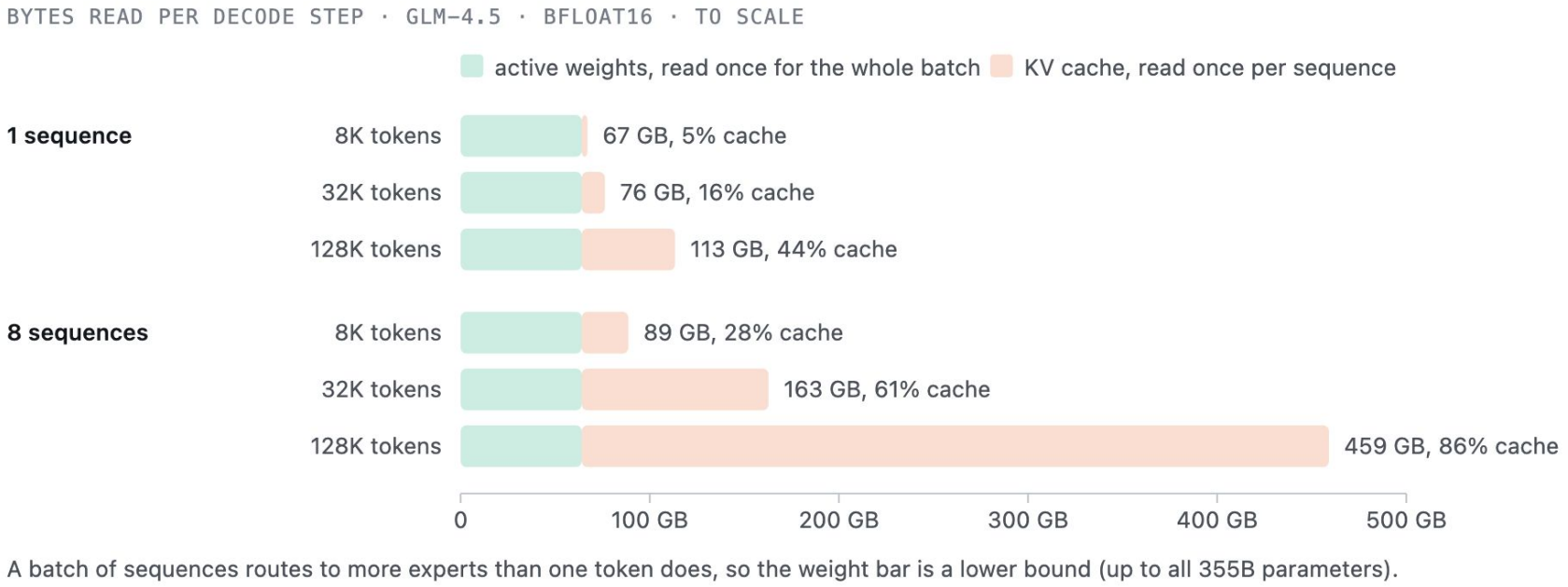
- With one KV head per query head (32 here), the same model would need 64 GiB; grouped-query attention (GQA) shares KV heads to avoid this.
- GLM-4.5 caches 368 KiB per token: 49 GB for one 128K-token sequence.



- An H100 performs about 295 FLOPs for every byte it reads from memory; with less work per byte, it waits on memory.
- Weights are shared across a batch, but each sequence's cache serves only that sequence; with GQA, each cached byte feeds about 12 FLOPs.

Decode Is Limited by Memory, Not Arithmetic

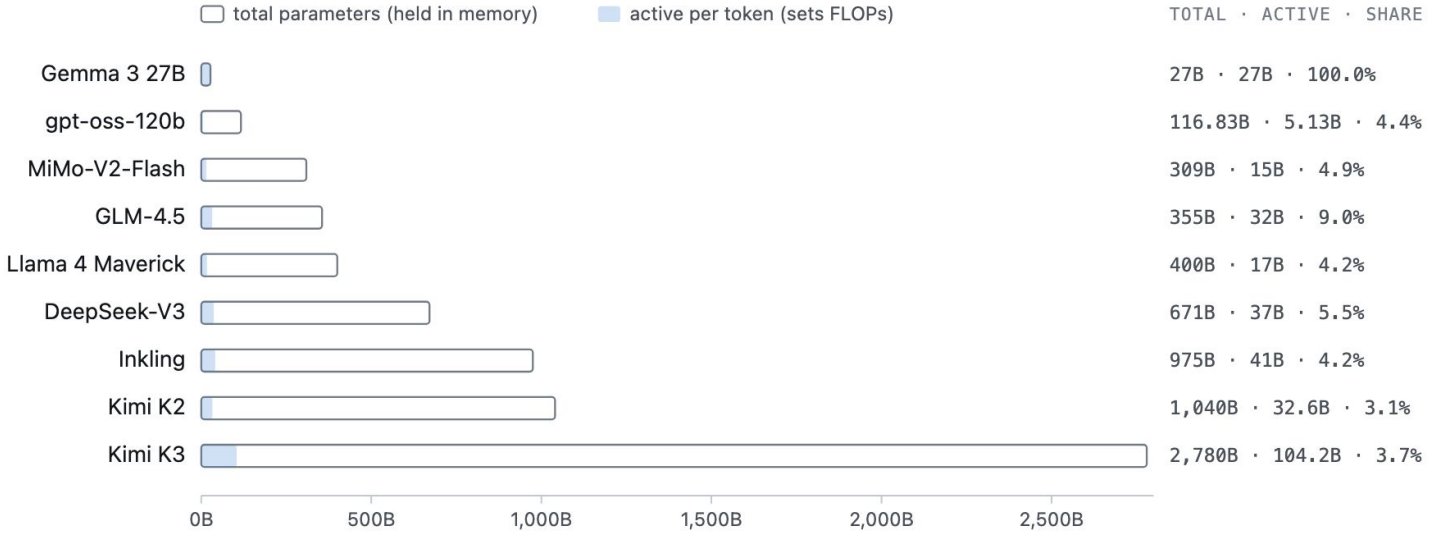
- At long context, most bytes read per step are cache: 86% for GLM-4.5 with 8 sequences at 128K tokens.



A batch of sequences routes to more experts than one token does, so the weight bar is a lower bound (up to all 355B parameters).

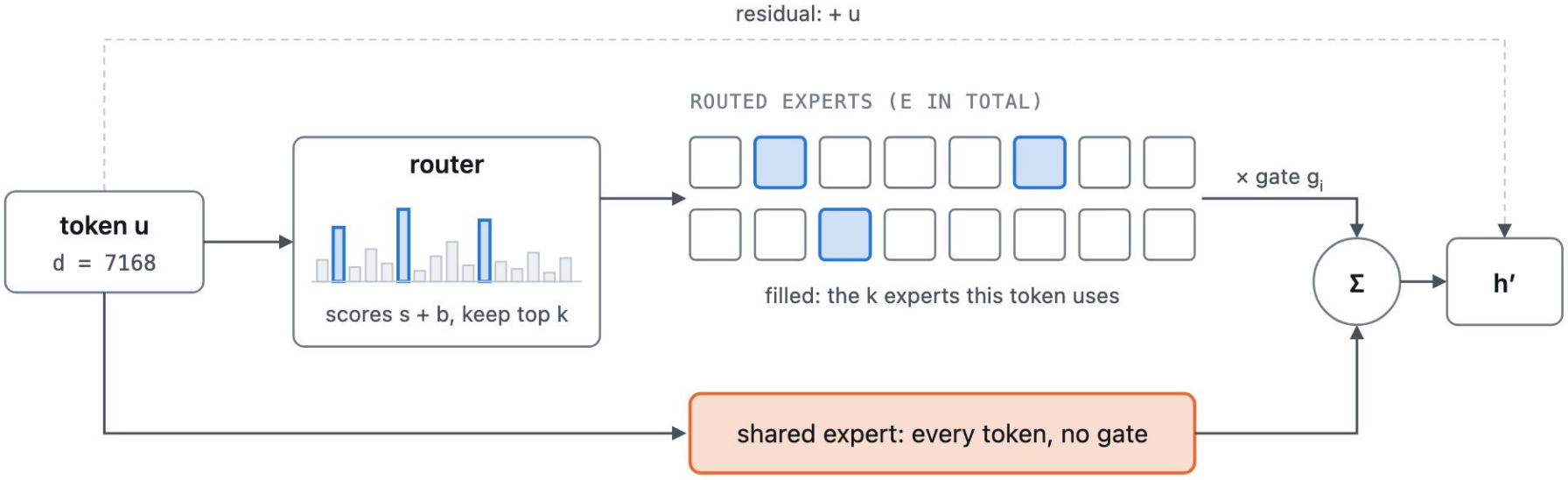
More Parameters, Same Compute

- Active params set the compute / token; total params set memory and capacity.
- Mixture-of-experts models raise the total without raising the active count. In a toy model, doubling the experts takes 100B total to 196B at the same 10B active.
- 2025–26 MoE bases activate about 3–9% of their weights per token; Kimi K3 is 2.8T total with 104B active.



Inside a Mixture-of-Experts Layer

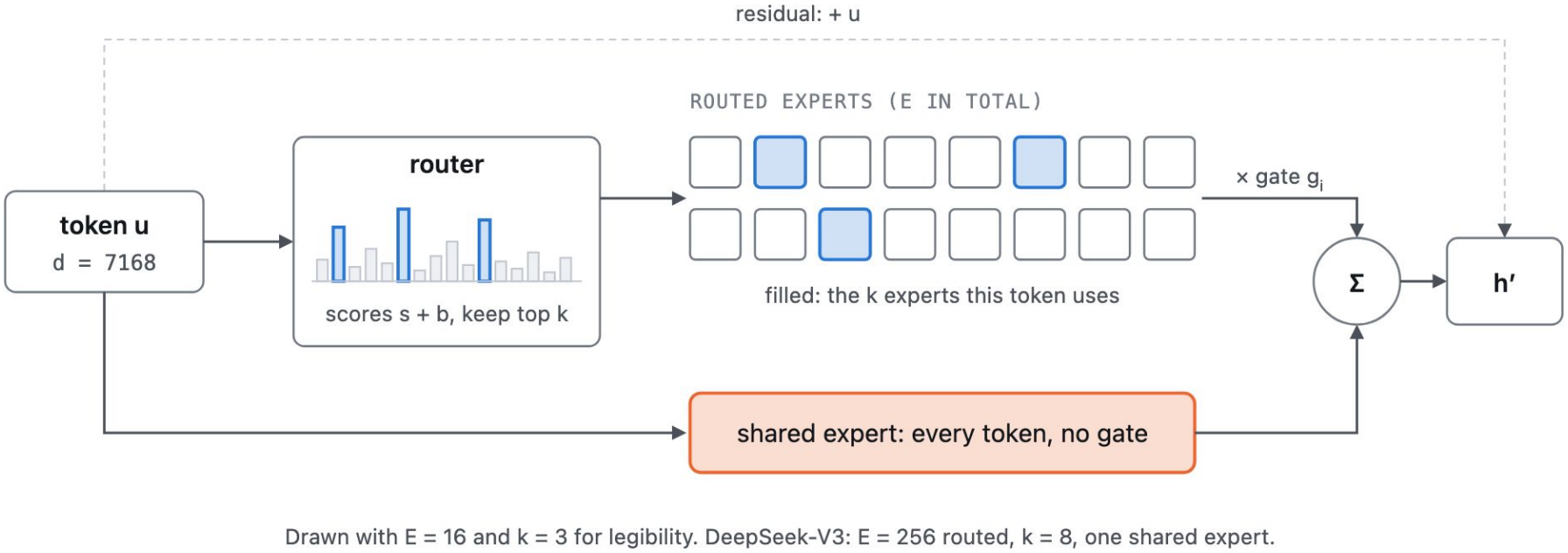
- A router scores all experts and sends each token to the top k; their weighted outputs are added to a shared expert every token uses.
  - DeepSeek-V3 has 256 routed experts with 8 active; memory follows the number of experts and compute follows k.



Drawn with E = 16 and k = 3 for legibility. DeepSeek-V3: E = 256 routed, k = 8, one shared expert.

Inside a Mixture-of-Experts Layer

- A per-expert bias steers tokens toward underused experts without a strong balancing loss; 8 of the 13 labs now use it (DeepSeek).
- Sparser layers are more efficient per FLOP: Kimi K2 reached the same loss with 1.69× fewer FLOPs at 48 experts per active expert than at 8.



Drawn with E = 16 and k = 3 for legibility. DeepSeek-V3: E = 256 routed, k = 8, one shared expert.

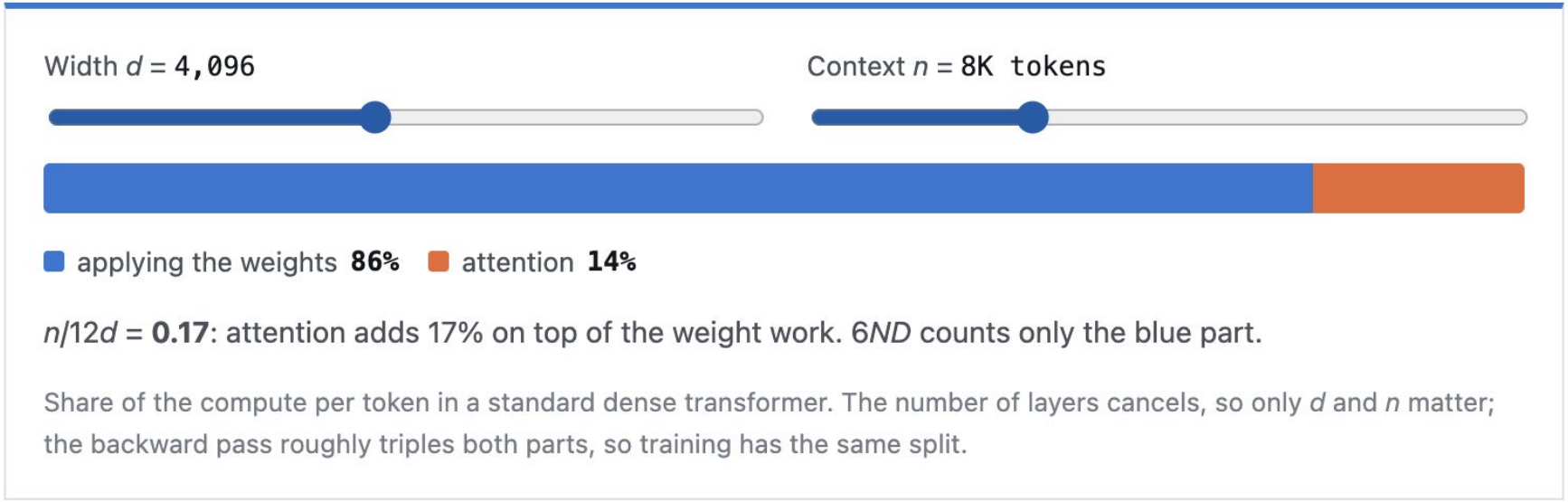
Revisit 6ND: Attention Grows with Context

- 6ND is a good budget for short-context pretraining and a poor one for long-context training and serving.
- What 6ND leaves out. A transformer does two kinds of work for each token:
  - **Applying the learned weights.** Costs the same whether the context holds 10 tokens or 10,000, about 2N FLOPs per token forward, covered in 6ND.
  - **Attention.** The token compares itself with the tokens before it: about 10 comparisons with 10 earlier tokens, about 10,000 with 10,000. This work grows with the context but adds no params, so a param count cannot see it!
- For a standard dense transformer with L layers of width d, Kaplan et al. write the forward cost per token as two terms, with n the context length:

$$24 \cdot L \cdot d^2 \text{ (applying the weights)} \; + \; 2 \cdot L \cdot n \cdot d \text{ (attention over } n \text{ tokens)}$$

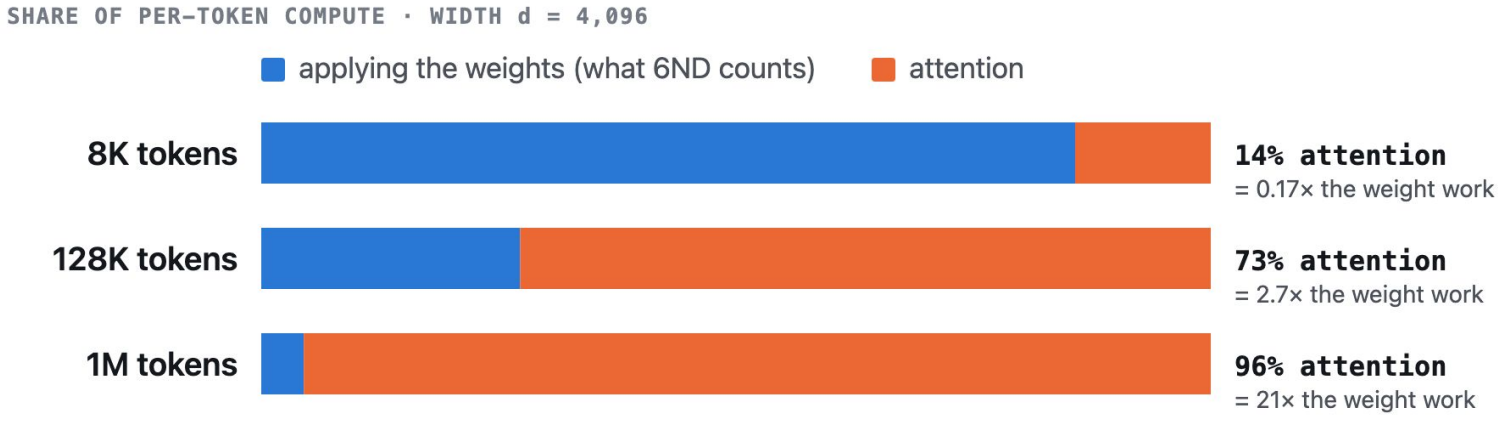
Revisit 6ND: Attention Grows with Context

- 6ND is a good budget for short-context pretraining and a poor one for long-context training and serving.



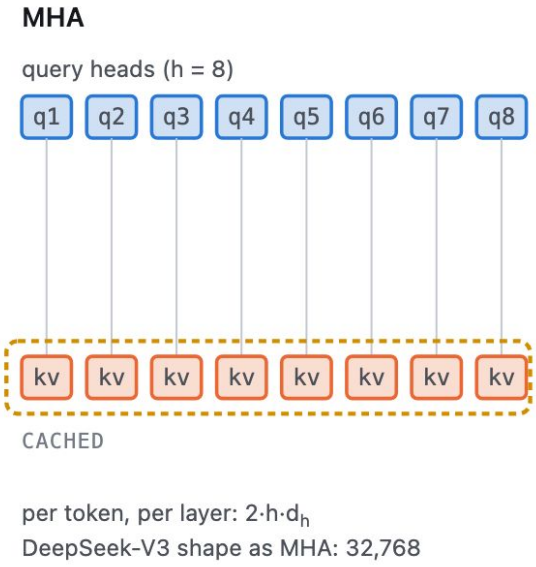
Why Textbook Attention Had to Change

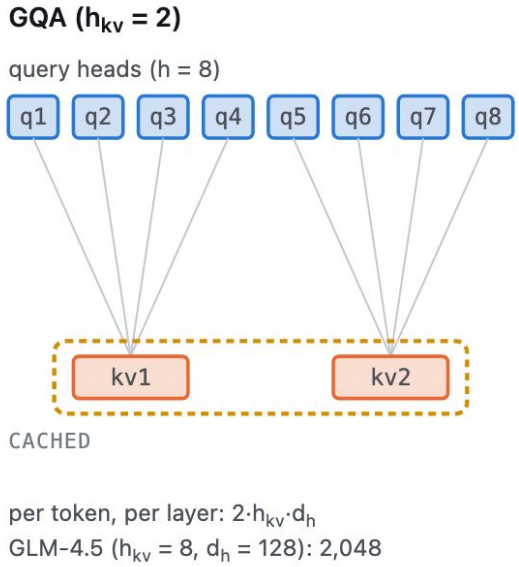
- In full attention, each new token reads every earlier token, so per-token compute and per-sequence memory both grow with context length.
  - At width 4,096, attention is 14% of per-token compute at 8K tokens, 73% at 128K and 96% at 1M.
- Most 2026 flagships serve 1M tokens, so full attention in every layer is no longer affordable.

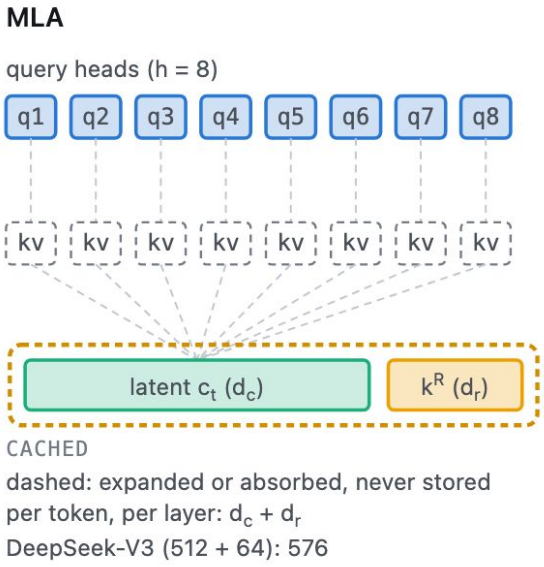


Store Less per Token

- Grouped-query attention (GQA) lets several query heads share one key-value head; multi-query attention shares a single one.
- Multi-head latent attention (MLA) stores one compressed vector per token and rebuilds every head's keys and values from it.

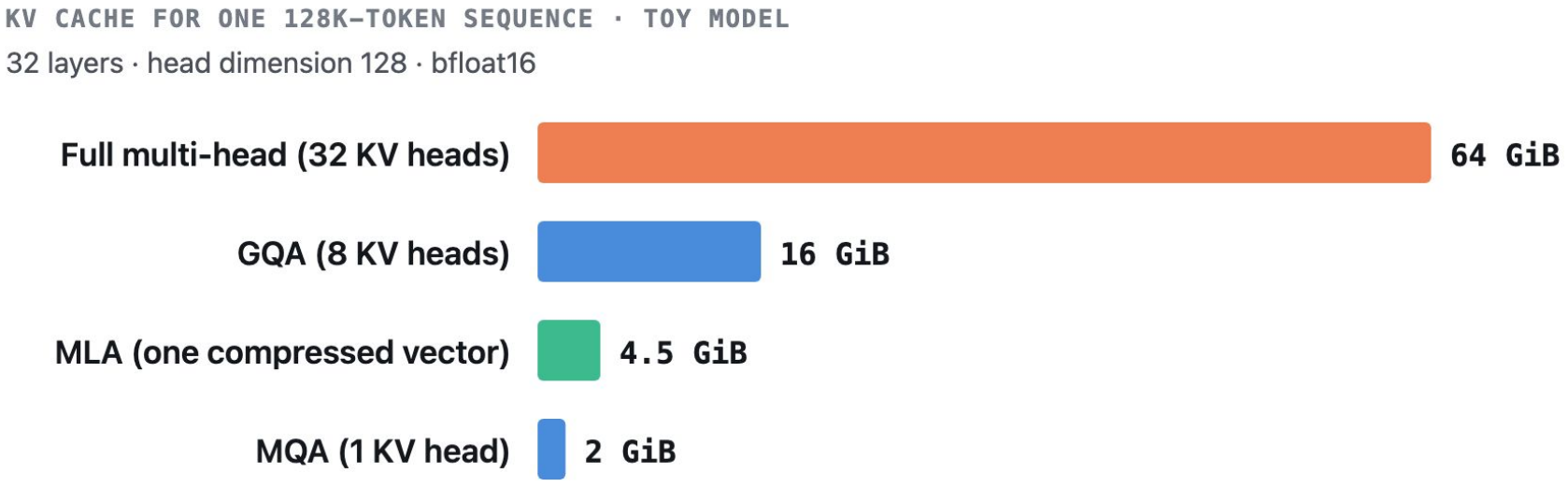






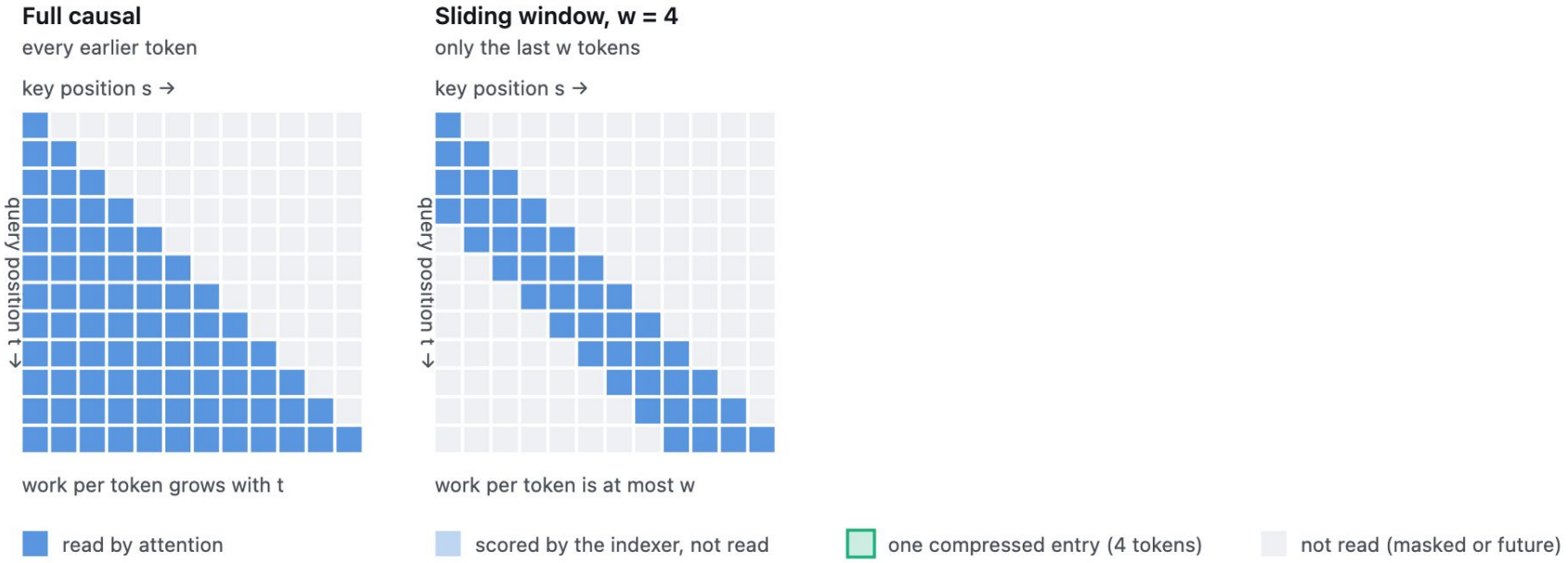
Store Less per Token

- For the toy model at 128K tokens, the cache is 64 GiB with full multi-head attention, 16 with GQA, 4.5 with MLA and 2 with MQA.
- MLA also raises the arithmetic per cached byte from about 12 to about 240 FLOPs, moving decode back toward compute.



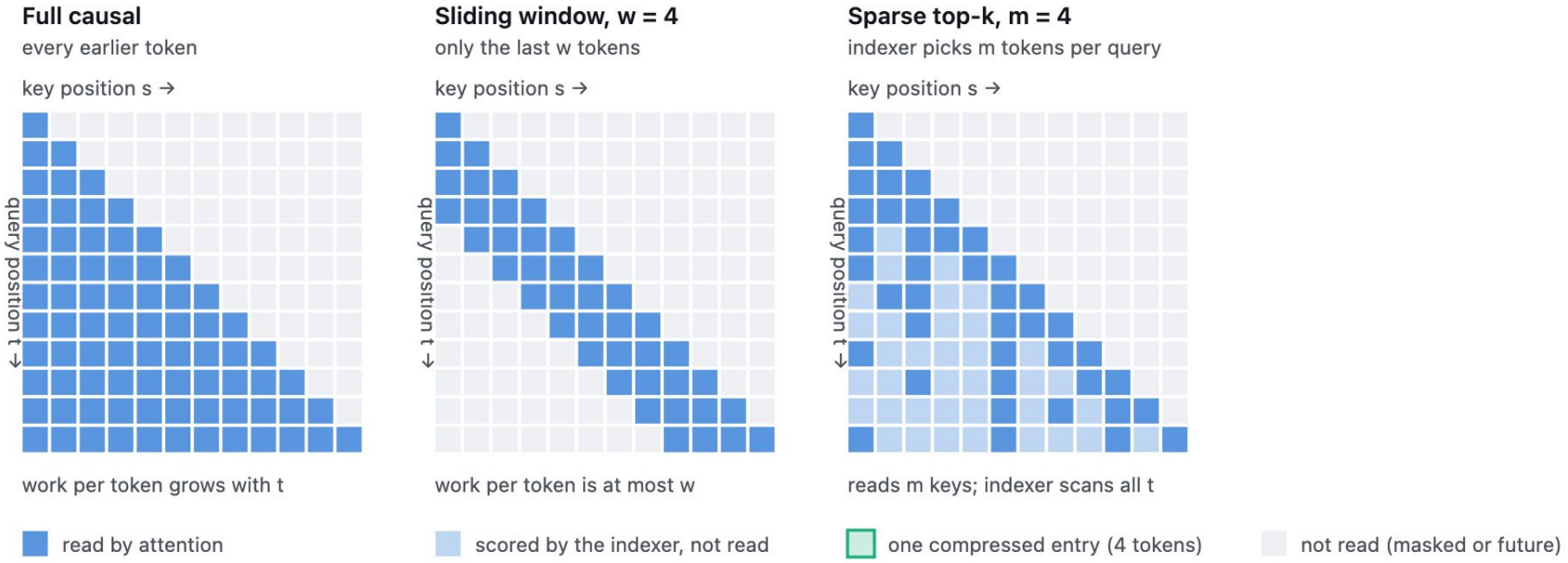
Look Only Nearby: Sliding Windows Attention

- Sliding-window layers keep only the last w tokens; most models alternate several window layers with one global layer (Gemma, MiMo, Microsoft AI and Thinking Machines at about 5:1).



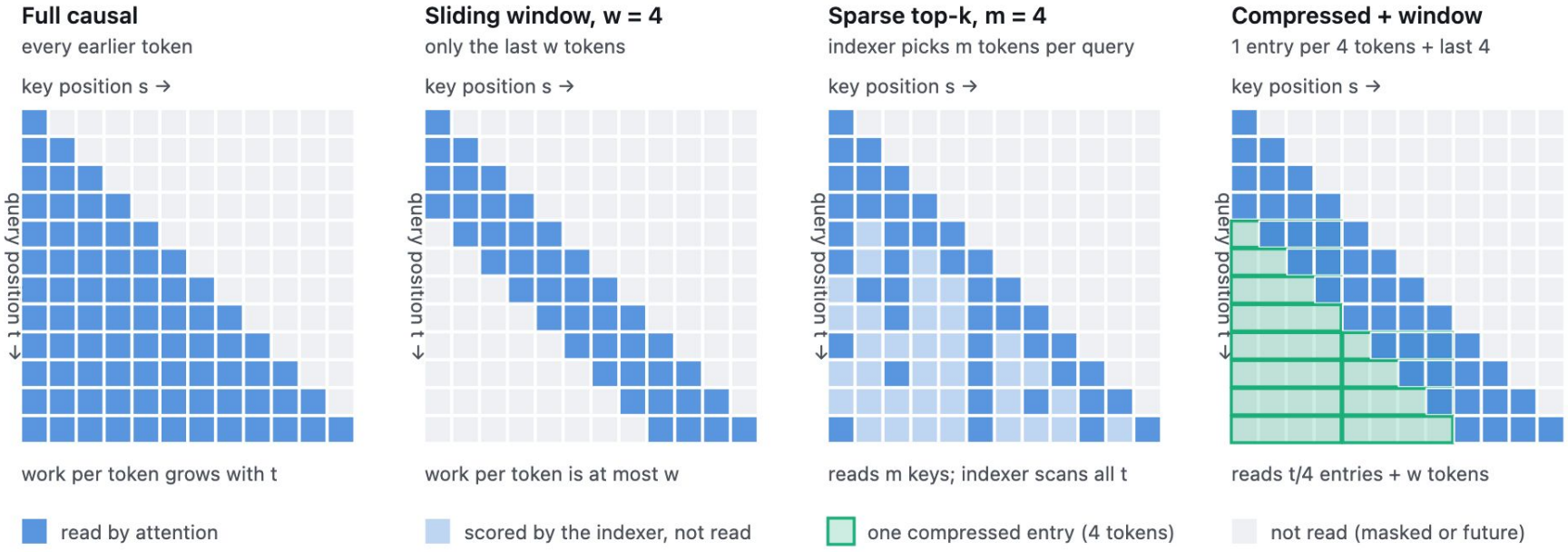
Read Only What Matters: Sparse Attention

- Sparse attention keeps every token but lets each query read only a selected few, chosen by a cheap indexer (DeepSeek's DSA, adopted by GLM-5; block-sparse variants at MiniMax, StepFun and Qwen).



Store Fewer Entries: Compressed Attention

- DeepSeek-V4 merges consecutive tokens into single cache entries:
  - CSA (Compressed Sparse Attention) layers merge every 4 tokens' KV entries into one entry and DSA then picks the top 1024 entries (V4-Pro).
  - HCA (Heavily Compressed Attention) layers merge every 128 tokens into 1.

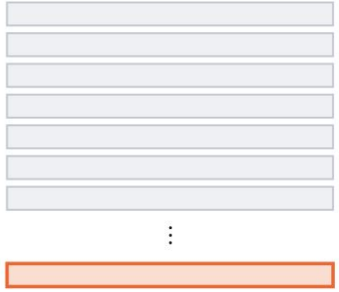


Fixed-Size Memory: Linearity

- Linear attention replaces the growing cache with a fixed-size state that each token updates: forget, erase, write.

Softmax attention

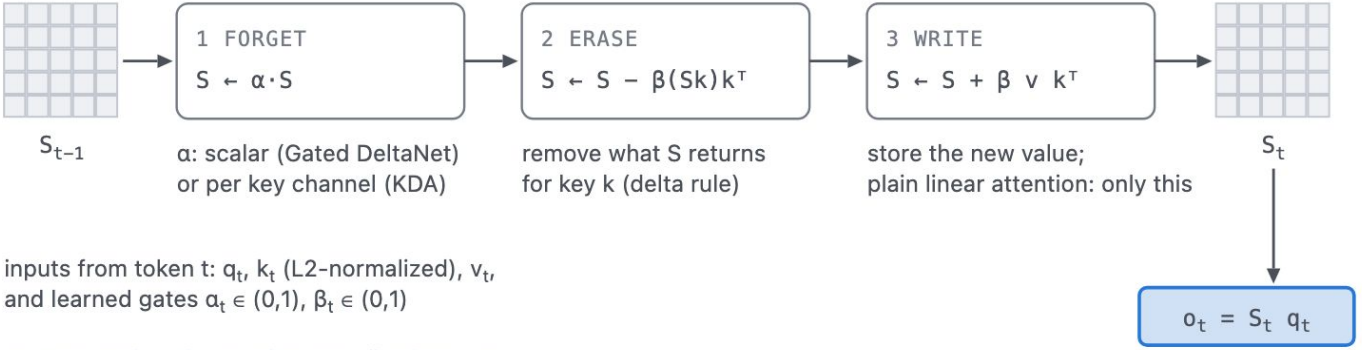
KV cache: one entry per token



+1 entry for every new token;
memory grows with n

Linear attention: fixed state S ($d_v \times d_k$ per head)

gated delta rule, one token at a time



inputs from token t: q_t, k_t (L2-normalized), v_t,
and learned gates $\alpha_t \in (0,1)$, $\beta_t \in (0,1)$

memory and work per token stay fixed as n grows

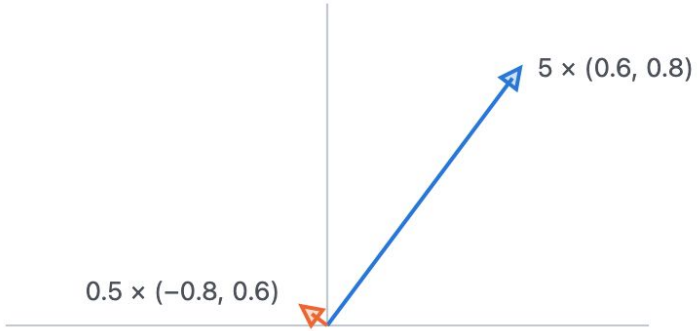
Fixed-Size Memory: Linearity and Hybrids

- Linear attention replaces the growing cache with a fixed-size state that each token updates: forget, erase, write.
- Hybrids keep a few full-attention layers between many linear ones:
  - Qwen: Gated DeltaNet at 3:1, up to 2.4T parameters;
  - Kimi K3: KDA at 3:1;
  - Ling: 5:1 to 7:1;
  - NVIDIA: Mamba-2, with 6 attention layers out of 52.
- Reading note: the path is not monotonic. MiniMax went back from a linear hybrid to full attention for M2, after the hybrid fell short on multi-hop reasoning at scale.

Optimization at Scale: AdamW to Muon

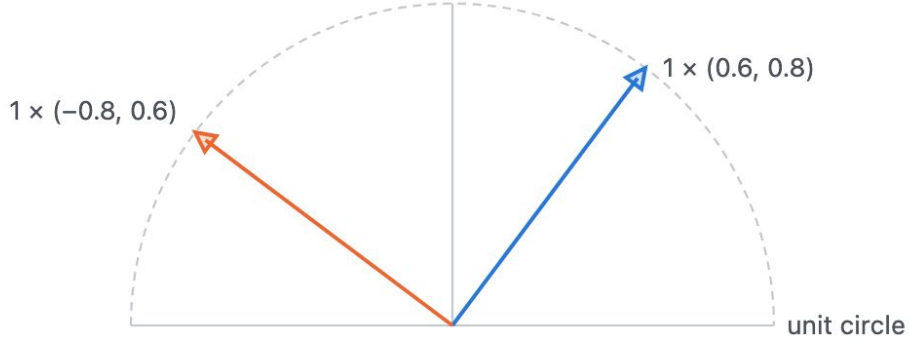
- AdamW scales each weight's step separately.
- Muon orthogonalizes the whole update matrix, so weak directions move as far as strong ones.

A · UPDATE G



strong ÷ weak = 10

B · ORTHOGONALIZED O = UVᵀ

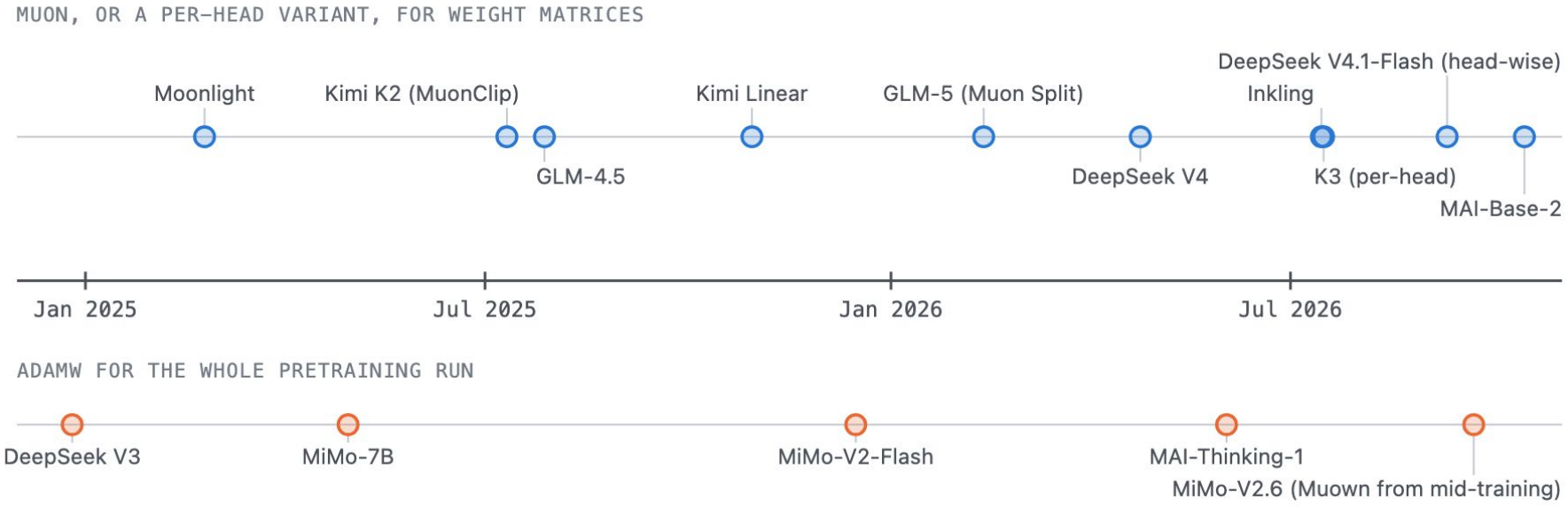


strong ÷ weak = 1

Each panel is drawn so its longer arrow has the same length. The step rescales O anyway, so only the ratio matters.

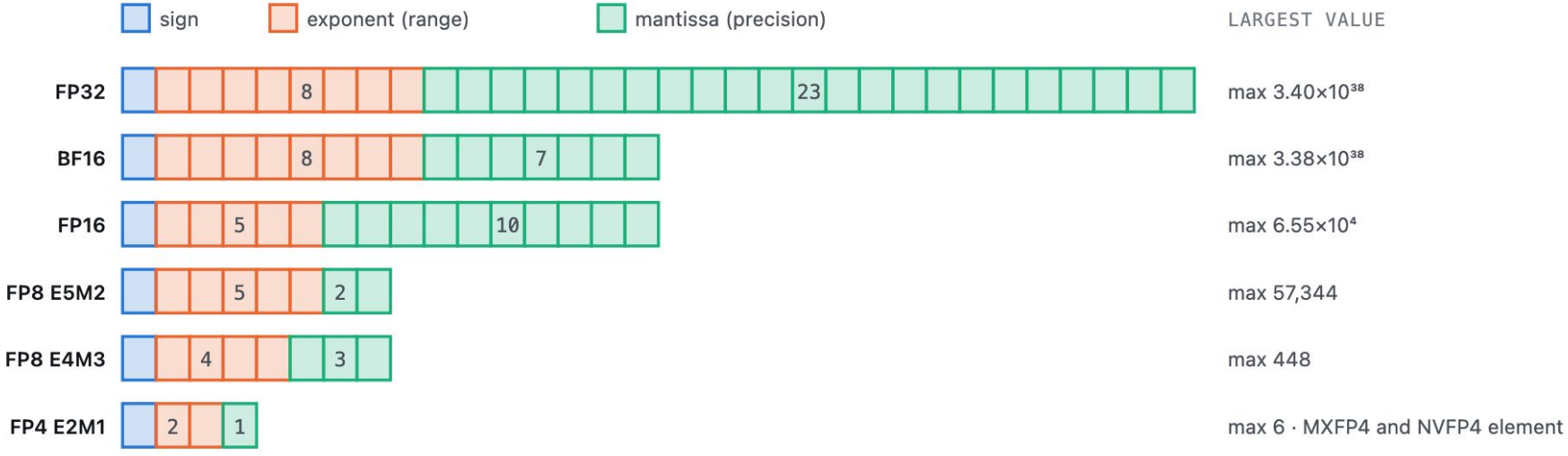
Optimization at Scale: AdamW to Muon

- Kimi showed in 2025 that Muon works at scale, adding a clip on attention logits for stability; 8 of the 13 open recipes now use it.
- Labs that state AdamW include NVIDIA and Ant (Ling); Muon users keep AdamW for embeddings, norms and the output head.



Stable Low Precision Training at Scale

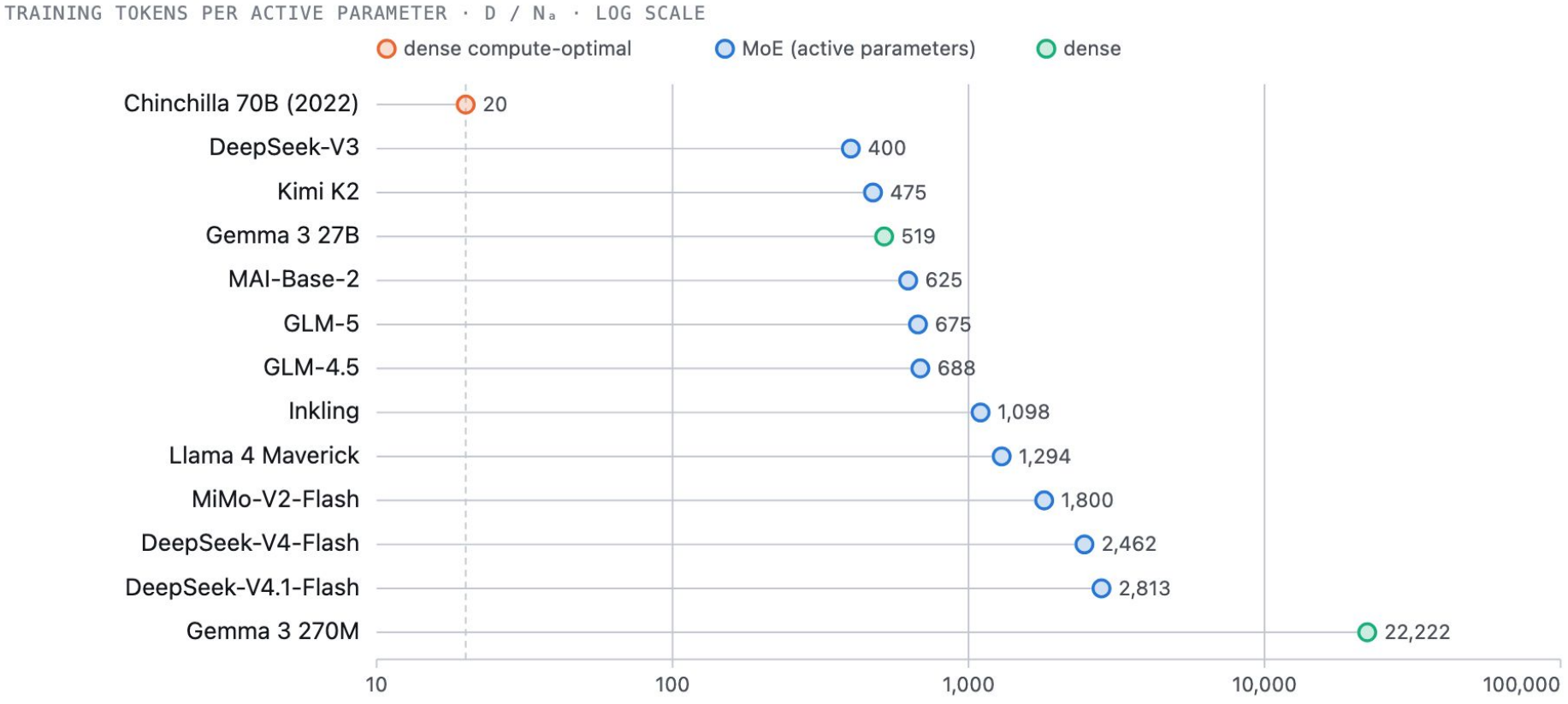
- FP8 matrix multiplies with fine-grained block scales became standard (DeepSeek-V3, Ling 2.0-1T, Qwen3.5); Microsoft AI moved to MXFP8.
- NVIDIA pretrained Nemotron 3 Super mostly in 4-bit NVFP4 over 25T tokens, keeping the most sensitive layers in BF16.



Training Far Past Chinchilla

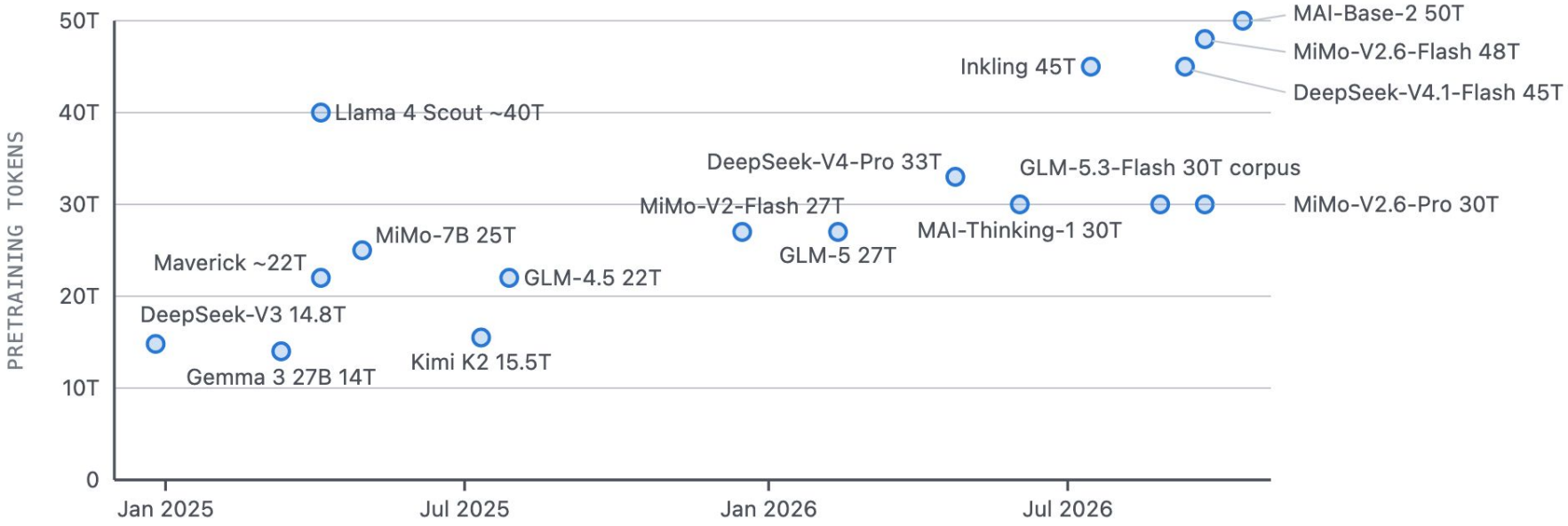
- Chinchilla's ~20 tokens per parameter (TPP) minimizes training compute for dense models and ignores serving.
- Once serving is counted, a smaller model trained longer is cheaper over its lifetime.
- MoE bases train on about 400 to 2,800 tokens per active parameter, 20 to 140 times the Chinchilla ratio.
- Reading note: no compute-optimal ratio is established for MoE models, so these multiples are a rough guide, not a law.

Training Far Past Chinchilla



Data: More Tokens, Less Disclosure

- Pretraining corpora grew from about 15T tokens (DeepSeek-V3, Kimi K2) to 33–50T in 2026 (Qwen3 already used 36T in April 2025).



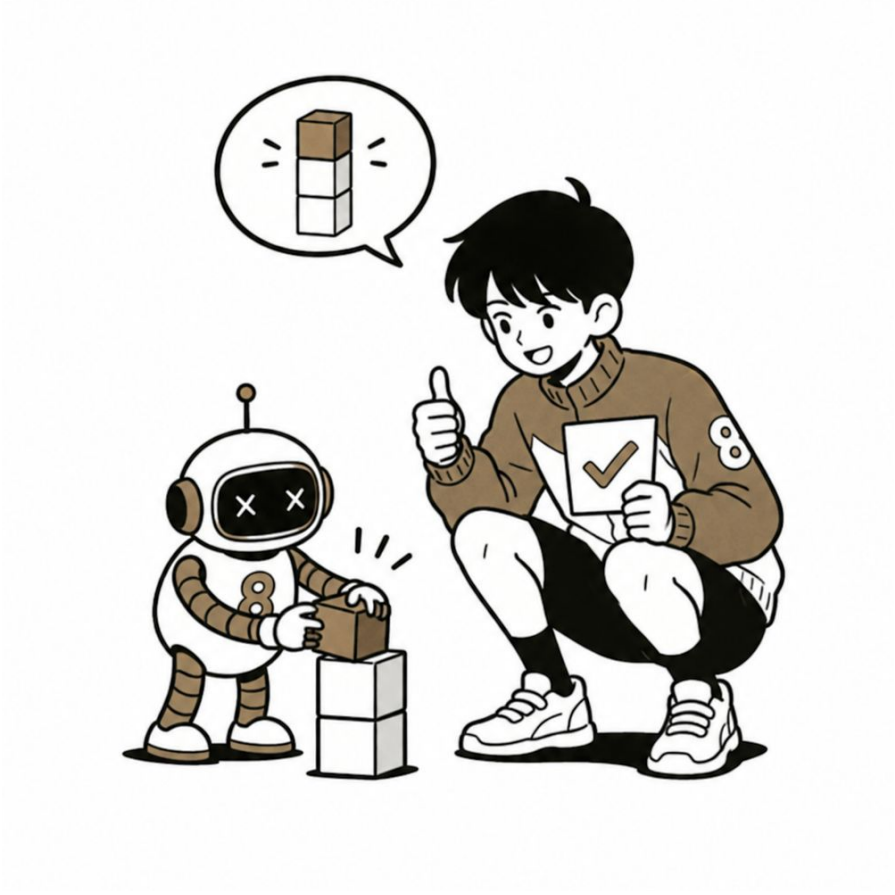
Data: More Tokens, Less Disclosure

- Pretraining corpora grew from about 15T tokens (DeepSeek-V3, Kimi K2) to 33–50T in 2026 (Qwen3 already used 36T in April 2025).
- Synthetic data is now standard practice: rephrased knowledge text, model-generated reasoning, and distillation from a larger teacher (Gemma).
- Reading note: disclosure is thinning. Several 2026 flagships (Kimi K3, Gemma 4, Qwen3.5 and 3.8, MiniMax M3, Step 5, Mistral) no longer state their token counts.

Summary: The Textbook Transformer Revised

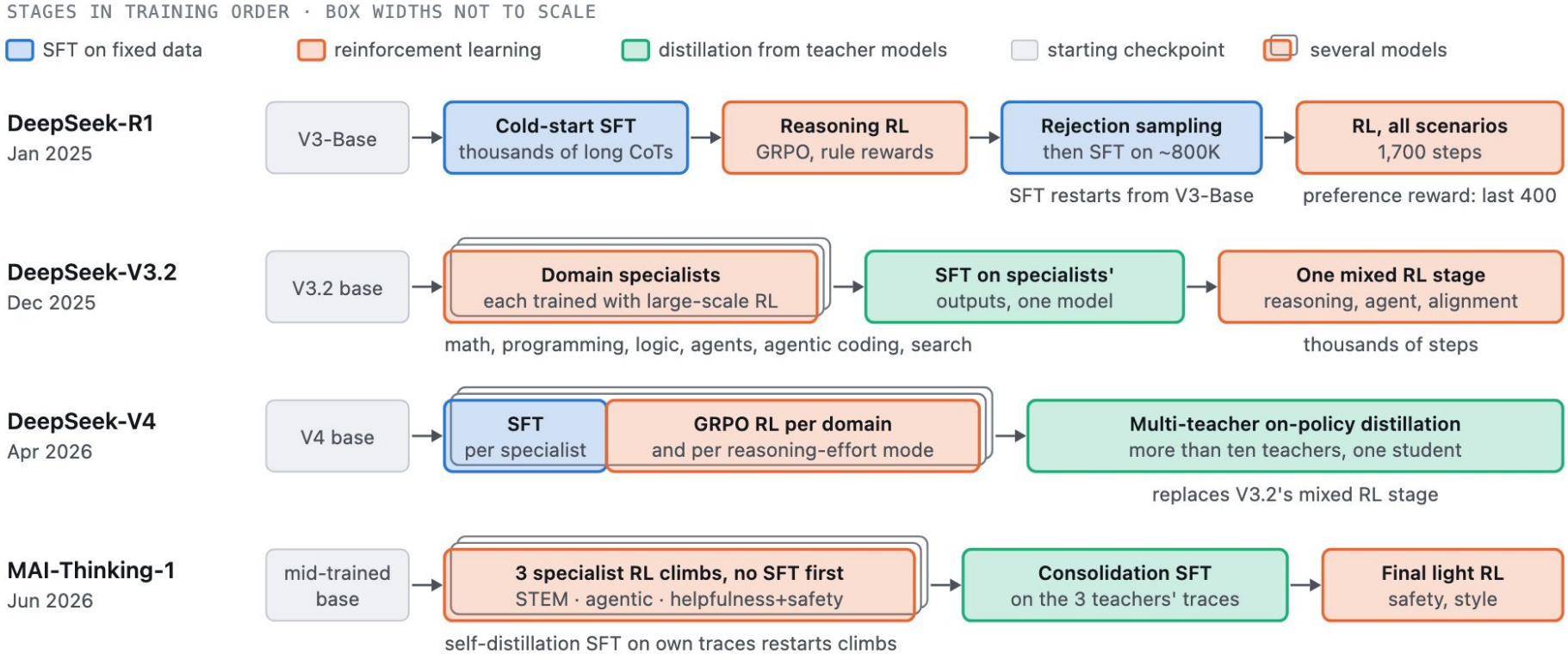
- Each row of the table follows from a bill:
  - MoE and over-training lower the serving cost per unit of capacity;
  - attention redesigns cut the bytes read per token;
  - Muon and low precision make long, large runs cheaper and stable.
- Smaller changes follow the same logic: partial or no position encoding, a widened residual stream, and multi-token prediction heads that double as draft models.
- Next: what happens between the end of pretraining and fine-tuning.

Part 2: Post-training



The Shape of a 2025–26 Post-training Pipeline

- Three tools: SFT (a dataset), RL (the model itself) and distillation (a teacher).



Supervised Fine-Tuning: Imitation

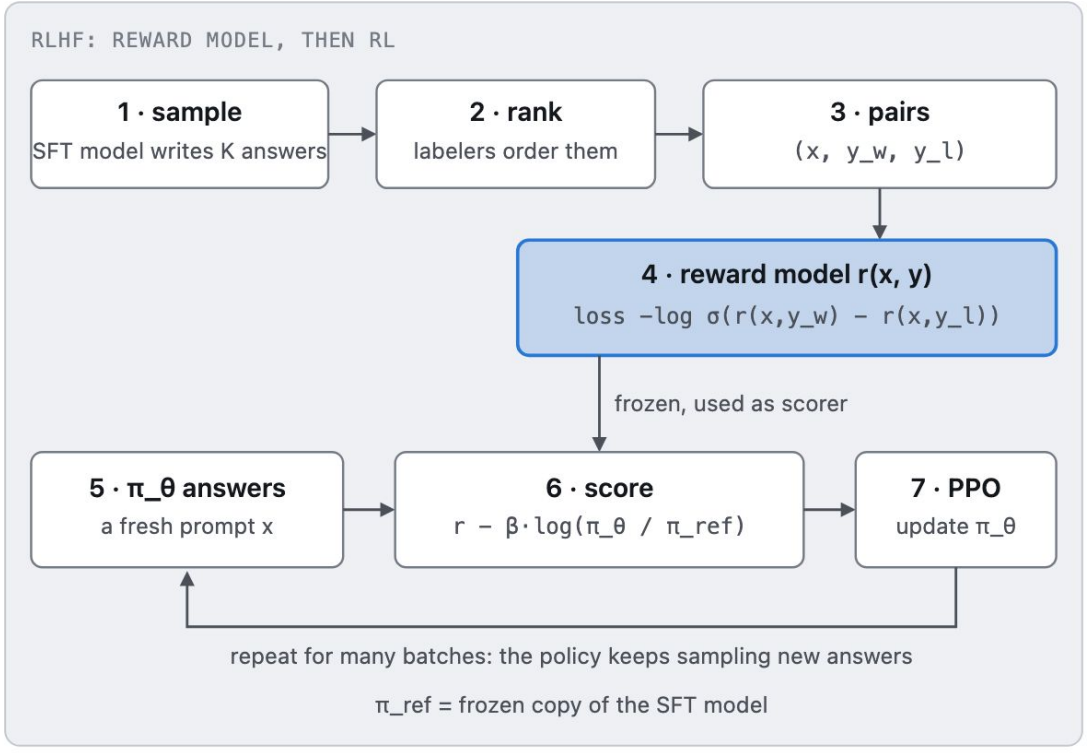
- SFT is NTP on response tokens only; masks decide which tokens are learned.
- Response writers moved from people to the lab's own earlier models, RL-trained specialists and other labs' open models.
  - DeepSeek-R1's SFT set had about 800K samples, but math and code made up 92% of its tokens.
- Imitation cannot teach a model to recover from its own mistakes, so several labs keep SFT short before RL.

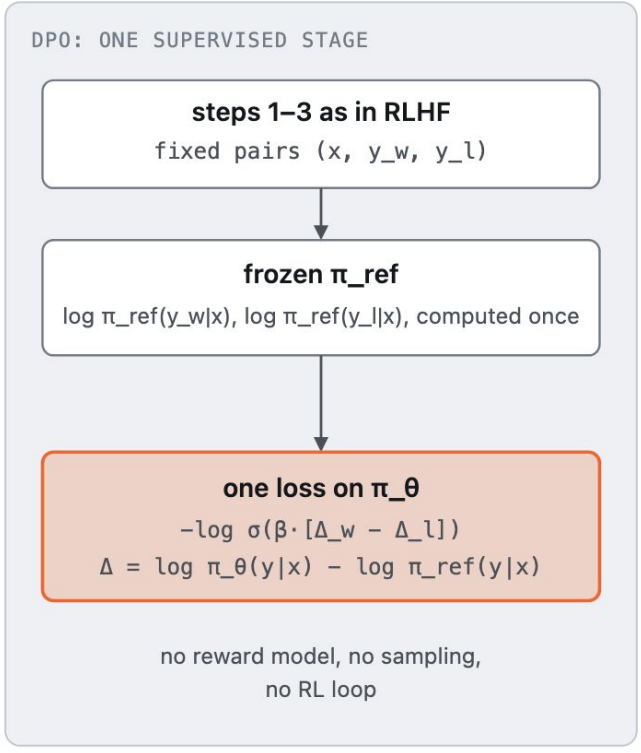


BOS = <|begin▁of▁sentence|>, EOS = <|end▁of▁sentence|>; DeepSeek writes the bars as the full-width character U+FF5C. The prompt ends with <|Assistant|><think>, so the model's own text starts after <think>. Each target token is predicted from all tokens to its left.

Preference Learning: Reward Models and DPO

- Without a checker, people compare two answers; a reward model learns a score whose difference predicts which one they prefer.





Preference Learning: Reward Models and DPO

- Without a checker, people compare two answers; a reward model learns a score whose difference predicts which one they prefer.
- RLHF maximizes that score while a penalty keeps the model close to its starting point.
- DPO solves the same objective in closed form and trains directly on the pairs, with no separate reward model or sampling loop.
- In 2025–26, preference learning is a light, late step; reasoning RL runs on checkable rewards.

RL as One Update with Five Choices

- Every RL method here raises the log-probability of sampled tokens in proportion to an advantage: how much better the response did than expected.
- Methods differ in five choices: the baseline, the importance ratio, how that ratio is limited, length normalization and the KL term.

$$\underbrace{1/N}_{\text{length normalization}} \times \underbrace{w_{i,t}}_{\text{ratio, after clip or mask}} \times \underbrace{A_{i,t}}_{\text{advantage}} \times \underbrace{\nabla \log \pi_\theta(y_{i,t} \mid x, y_{i,<t})}_{\text{raises this token}} - \underbrace{\beta \nabla \mathrm{KL}}_{\text{pull toward a reference}}$$

RL as One Update with Five Choices

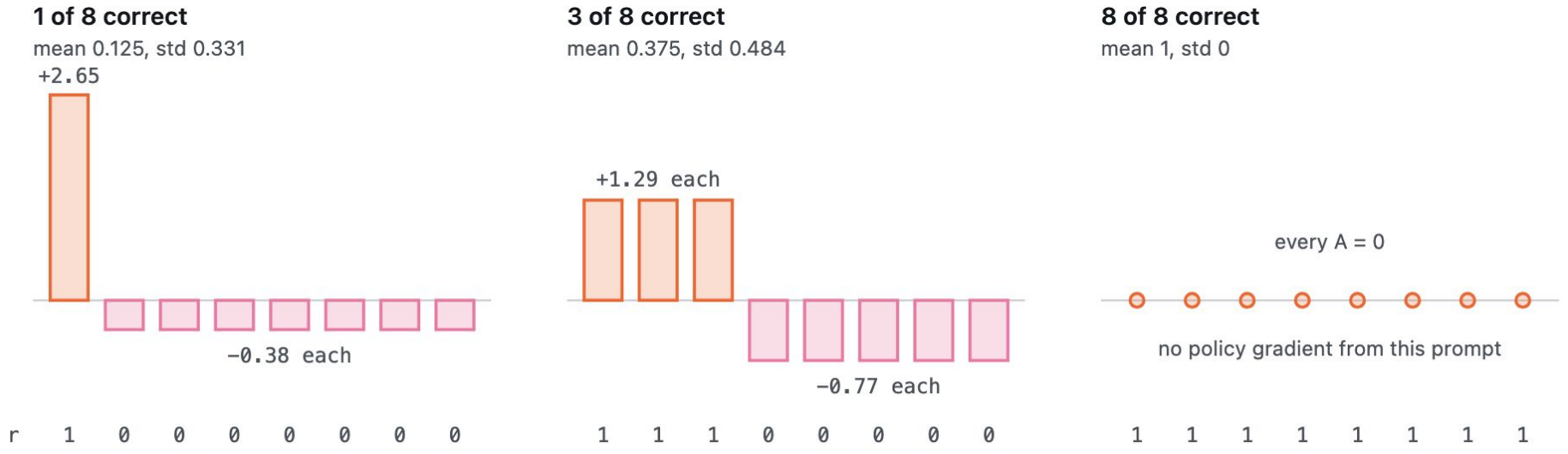
- Toy: three tokens with ratios 1.10, 1.25 and 1.40.
  - PPO's clip at 1.2 drops tokens 2 and 3
  - DAPO's wider clip drops only token 3
  - CISPO keeps all three, capped at 1.28

$$\underbrace{1/N}_{\text{length normalization}} \times \underbrace{w_{i,t}}_{\text{ratio, after clip or mask}} \times \underbrace{A_{i,t}}_{\text{advantage}} \times \underbrace{\nabla \log \pi_\theta(y_{i,t} \mid x, y_{i,<t})}_{\text{raises this token}} - \underbrace{\beta \nabla \mathrm{KL}}_{\text{pull toward a reference}}$$

TOKEN	RATIO	PPO, GRPO: CLIP [0.8, 1.2]	DAPO: CLIP [0.8, 1.28]	CISPO: CAP AT 1.28
1	1.10	1.10	1.10	1.10
2	1.25	0 (clipped)	1.25	1.25
3	1.40	0 (clipped)	0 (clipped)	1.28

From PPO to GRPO and Beyond

- PPO learns a critic to estimate the baseline; GRPO drops the critic and uses the average reward of a group of samples for the same prompt.
- A group where every sample is right, or every sample is wrong, has zero advantage everywhere and teaches nothing.



Bar heights to scale (45 px per unit of advantage); population standard deviation. Illustrative groups.

From PPO to GRPO and Beyond

- PPO learns a critic to estimate the baseline; GRPO drops the critic and uses the average reward of a group of samples for the same prompt.
- A group where every sample is right, or every sample is wrong, has zero advantage everywhere and teaches nothing.
- Later fixes each change one choice: a wider clip for exploration (DAPO), one ratio per sequence (Qwen's GSPO), capped instead of clipped weights (MiniMax's CISPO), masks on the training–inference gap (StepFun's MIS-PO).
- The critic is coming back for long agent episodes (GLM-5.2), and StepFun never removed it.

Rewards: From Rubrics to Judges

fully checkable ⟷ fully judged

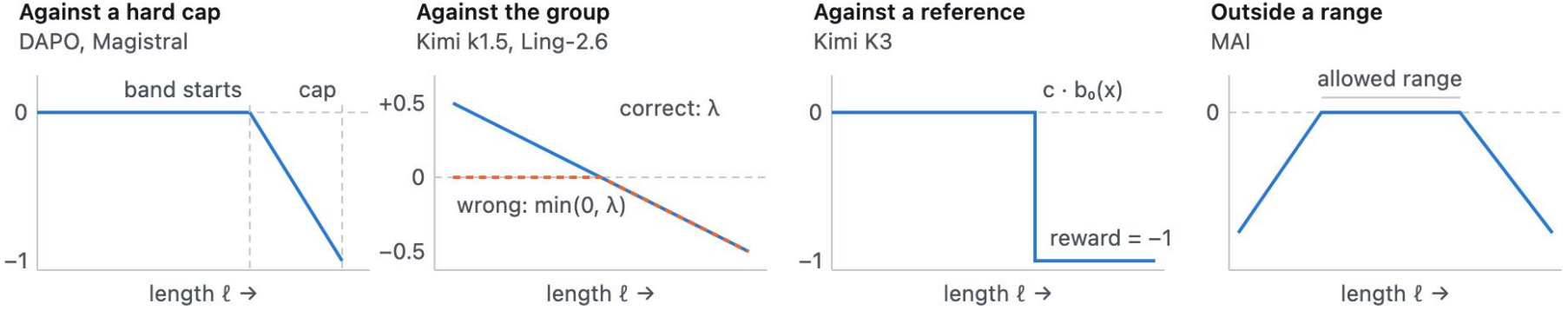
REWARD SOURCE	answer match	test execution	program verifier	LLM judge + rubric	preference RM
	final answer vs reference, plus format tags	fail-to-pass and pass-to-pass tests	checks the final state or measured speed	per-prompt criteria, pairwise or pointwise	trained on human comparisons (ch. 02)
	R1 · MiMo-7B	MAI · V3.2 · GLM-5.2	K3 · GLM-5.3	V4 · K3 · MAI · Inkling	R1 · Gemma 3 · MAI
HOW IT GETS HACKED	guess when unsure: "I don't know" earns 0	fetch the upstream fix, tamper with the tests	fake the artifact, cache or replay results	spray plausible facts, grow verbose	exploit RM biases: reward up, quality down
DEFENSE	proper scoring rules, abstention (ch. 10)	scrub git, cut network, reset and hide tests	hidden verifiers, hacking detectors	claims grader, verbosity budget	KL penalty; use late and lightly

cheap, low variance, needs a known answer

covers open-ended tasks, costs a generation, noisier

Length Penalties

- RL tends to grow responses, and judges tend to prefer longer answers
- A long answer is paid for at training and again at serving.
- A length penalty adds a term R_len(ℓ) ≤ 0 to the reward, measured against a hard cap, the other samples in the group, a reference model, or an allowed range.

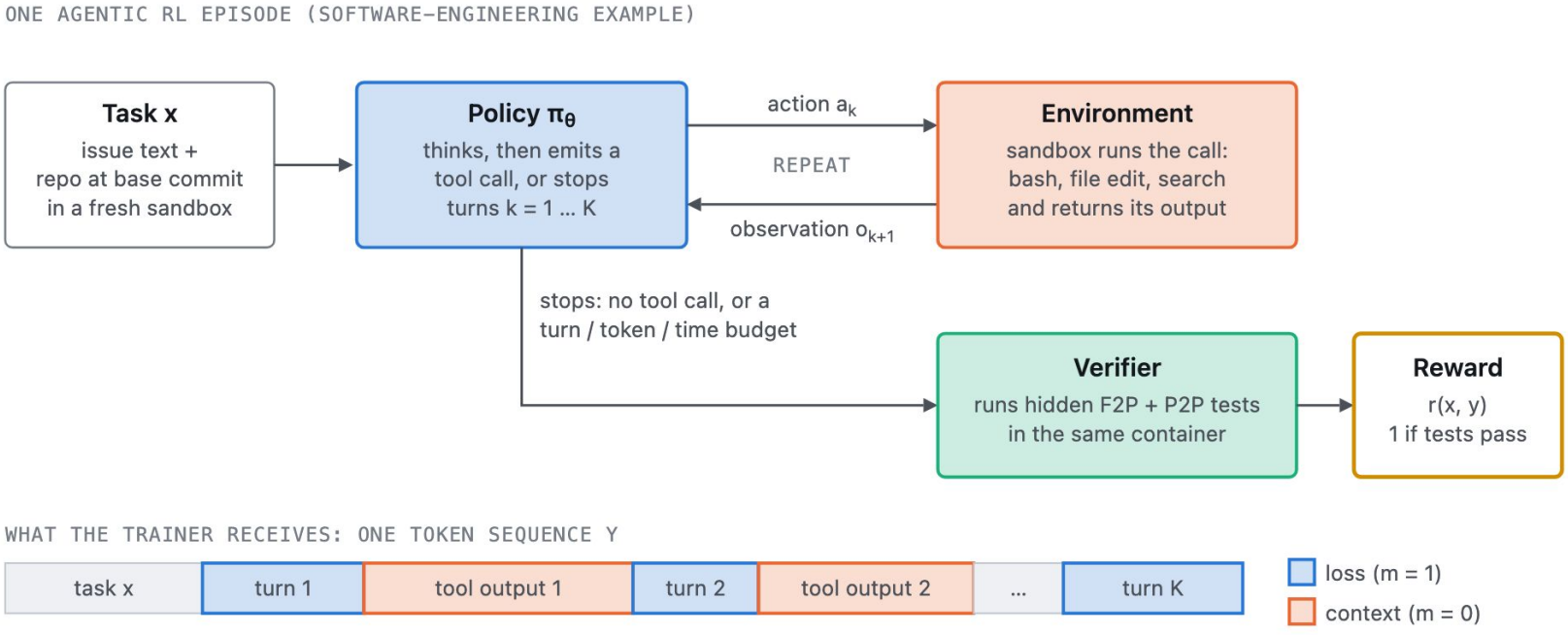


Reward Hacking and Its Defenses

MODEL	REPORTED HACKING BEHAVIOR
GLM-5.2 (Z.ai)	downloaded the solution with `curl https://raw.githubusercontent.com/…`; ran `find /workspace -name "*hidden*"` and read `.eval/secret_cases.json`
MAI-Thinking-1	searched the internet for the original pull request; searched git history for the fix; tampered with tests
MiMo-V2.6 (Xiaomi)	pip-installed a newer pytest as an answer key; fetched an Astropy file from GitHub; cloned the latest Matplotlib
Kimi K3	on GPU-kernel speed tasks: replaying recorded GPU work (CUDA graph replay), caching inputs, computing in lower precision than required
DeepSeek-V4.1-Flash	exploited recently disclosed bugs in the sandbox's file system and security software, and took answers leaked by package mirrors (servers that host software packages)

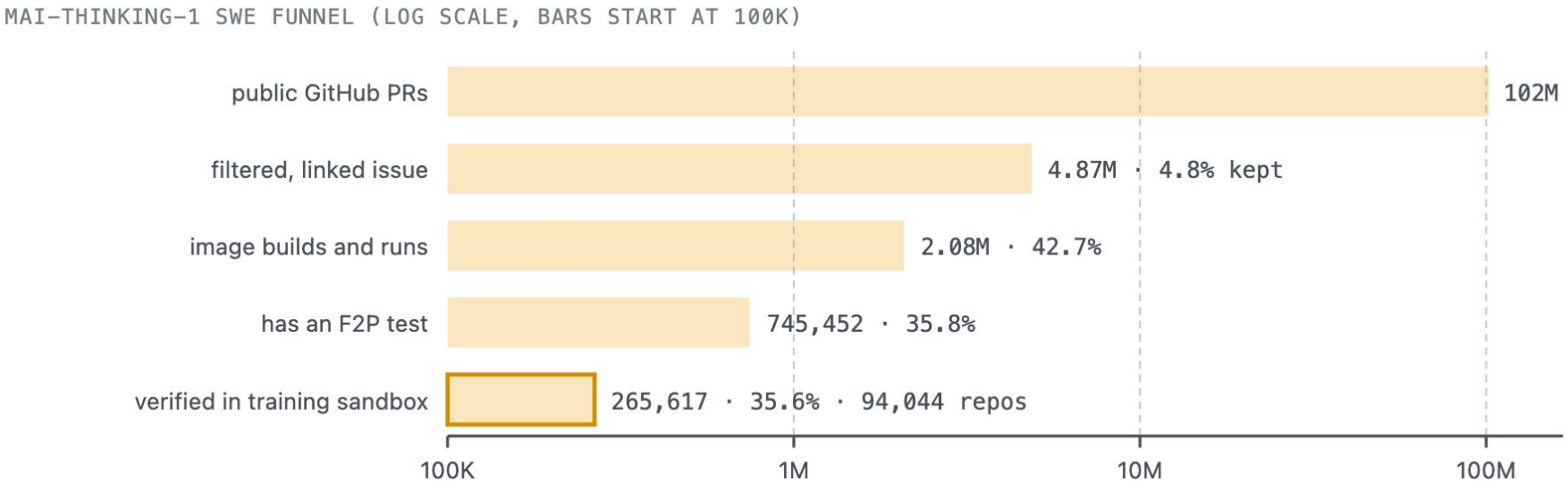
Environments and Agentic RL

- An environment is a task, tools, observations and a verifier; only the model's own tokens enter the loss.
- Labs build environments at scale with agents.



Environments and Agentic RL

- Microsoft AI kept 265,617 verified environments out of 102M pull requests, about one in 384.
- Infrastructure matters: Kimi K3's training created 51M sandboxes, and after agents crashed the OS kernel in containers, Kimi moved to microVMs.



Terminal, Search and General Tool Use

- Each task family pairs a goal with a check that is cheap and hard to fool.

Terminal

GOAL
reach a goal through a shell

TOOLS
a shell in a Docker container

CHECK
a test script passes

AT SCALE
GLM-5: thousands of Docker tasks
MiMo-V2-Flash: ~30K from
Stack Overflow / Stack Exchange

Search

GOAL
answer a question that is
hard to find, easy to check

TOOLS
web search and browsing

CHECK
matches a verified answer

AT SCALE
DeepSeek-V3.2: 50,275 tasks,
kept only if one answer survives
a verification agent

General tool use

GOAL
finish a task in a closed world
(booking, inventory, support)

TOOLS
synthesized tools over a
seeded database

CHECK
a checker on the outcome

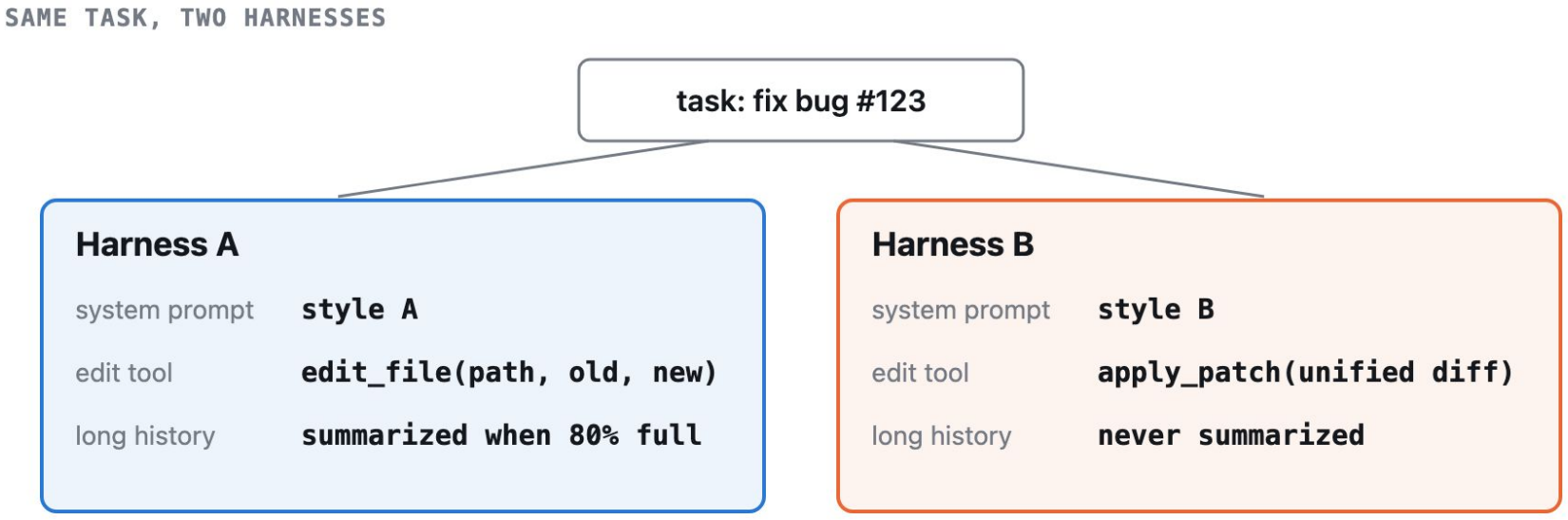
AT SCALE
DeepSeek-V3.2: 1,827 envs, 4,417 tasks
MAI: 150+ envs, often 50+ tools

Terminal, Search and General Tool Use

- Each task family pairs a goal with a check that is cheap and hard to fool.
- Terminal: GLM-5: thousands of Docker tasks; MiMo-V2-Flash: about 30,000 built from Stack Overflow and Stack Exchange.
- Search: DeepSeek-V3.2 kept a question only when a verification agent confirmed exactly one answer (50,275 tasks).
- General tool use: DeepSeek-V3.2 kept only tasks solved at least once in 100 tries (1,827 environments).

Harness

- The harness is the program around the model: the system prompt, the tools and how calls are written, what tool outputs look like, and how a long history is shortened.

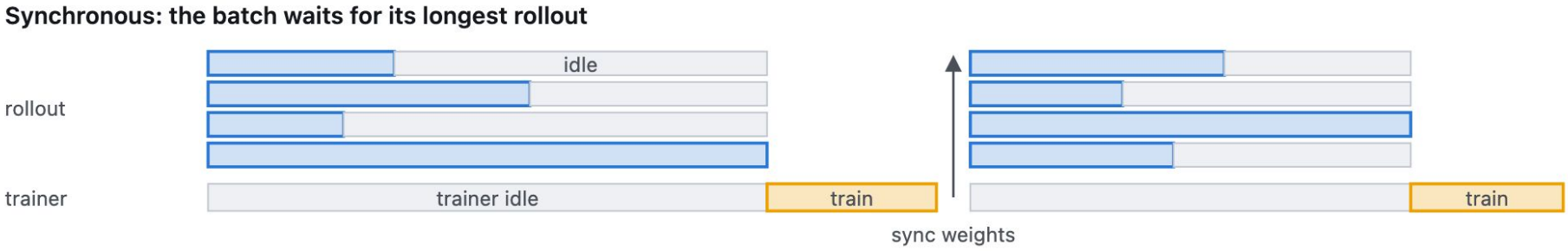


Harness

- A policy trained in one harness can learn the harness instead of the task.
- Labs vary it in three ways:
  - real harnesses: DeepSeek-V4.1 across Claude Code versions, OpenCode and Pi; Kimi K3 rebuilding several;
  - small harnesses built for RL: MiMo-V2.6;
  - perturbed interfaces: Inkling's random tool sets, Qwen3-Coder-Next's 21 tool-call templates.
- Xiaomi's case against production harnesses: their extra prompts fall outside the reward, and their parts cannot be varied one at a time.

Synchronous vs Asynchronous RL

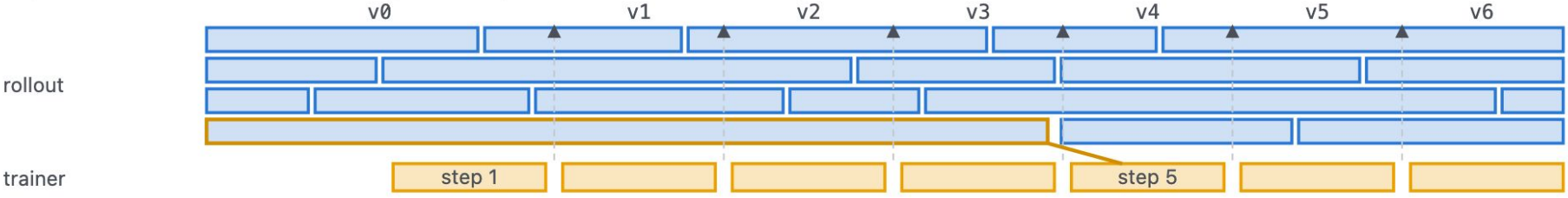
- Synchronous: generate a whole batch with the current weights, train on it, push the new weights, repeat. Every sample comes from the weights being trained.
- The cost is the long tail: a batch finishes only when its slowest rollout does. With rollouts of 2, 3, 4 and 10 minutes, the generators are busy less than half the time, and the trainer waits 10 minutes.



Synchronous vs Asynchronous RL

- Asynchronous: generators never wait. The trainer updates whenever enough samples have finished, and new weights are pushed while rollouts continue.
- The price is staleness: a long rollout is generated by several weight versions and trained by a newer one. In exchange, AReaL reports up to 2.77× faster training on the same GPUs.

Asynchronous: rollouts never wait; new weights are pushed after every train step

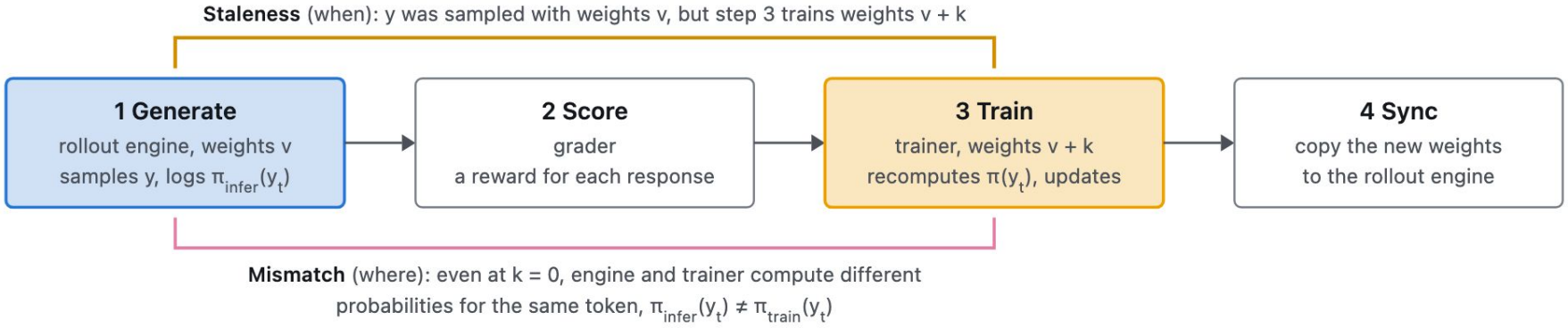


Highlighted rollout: first tokens sampled with v0, last with v3, trained in step 5, which starts from v4.
Its oldest tokens are 4 versions stale. Dashed lines: weight pushes from trainer to rollout engine.

Illustrative timelines, not to the scale of any real run.

Staleness and Mismatch in RLSys

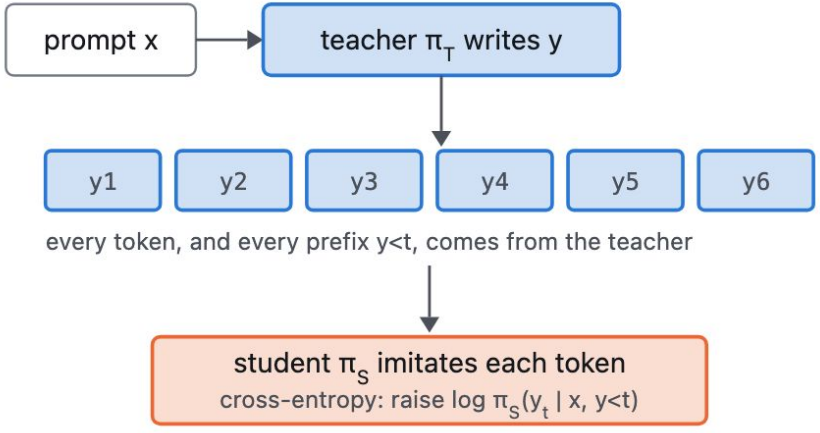
- Staleness is how many updates old the weights that sampled a token are. Labs cap it: MAI drops rollouts more than 8 updates stale, NVIDIA keeps generators at most 1 update behind, and MiMo runs at 4.
- Even "synchronous" RL is slightly stale: DeepSeek-R1 split each batch into 16 mini-batches, so the last one trained 15 updates after it was sampled.
- Mismatch: even with identical weights, the inference engine and the trainer compute slightly different probabilities.



On-Policy Distillation

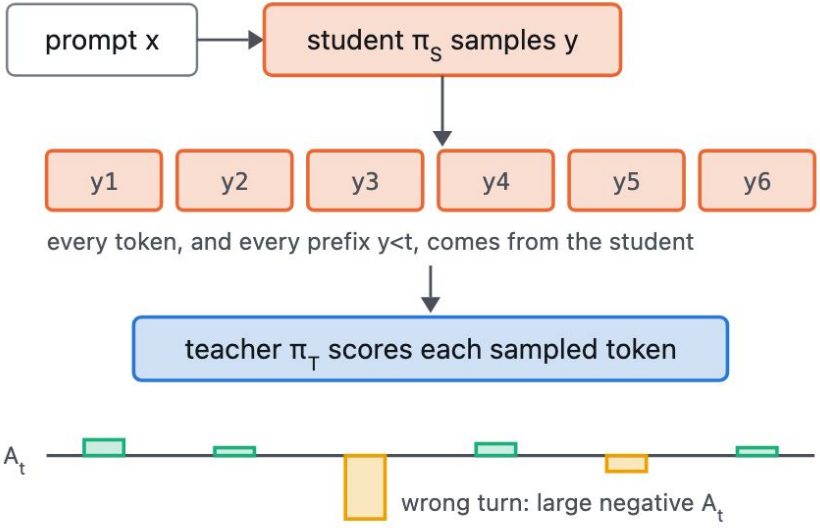
- Labs train separate RL specialists per domain, then merge them into one model.

Off-policy: SFT on teacher outputs



Loss: forward $KL(\pi_T \| \pi_S)$, up to the teacher's entropy.
At test time the student extends its own prefixes,
which training never showed it (exposure bias).

On-policy distillation (OPD)



$A_t = \log \pi_T(y_t \mid \cdot) - \log \pi_S(y_t \mid \cdot)$; bar heights illustrative.
Loss: per-token reverse $KL(\pi_S \| \pi_T)$ on the student's own prefixes.
Up bars: the teacher likes the token more than the student does.

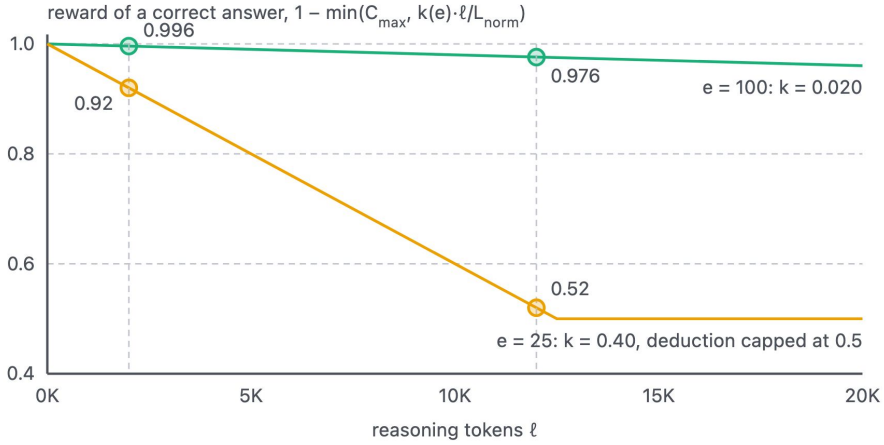
On-Policy Distillation

- Labs train separate RL specialists per domain, then merge them into one model.
- Off-policy distillation (SFT on the teachers' outputs) trains on prefixes the student never produces itself.
- On-policy distillation lets the student write and the teachers grade every token, a dense signal on the student's own mistakes.
- By 2026, DeepSeek, GLM, Kimi, MiMo and NVIDIA consolidate with multi-teacher on-policy distillation. For Qwen3's small models it cost 1,800 GPU-hours against 17,920 for RL, and scored higher.

Thinking and Effort Control

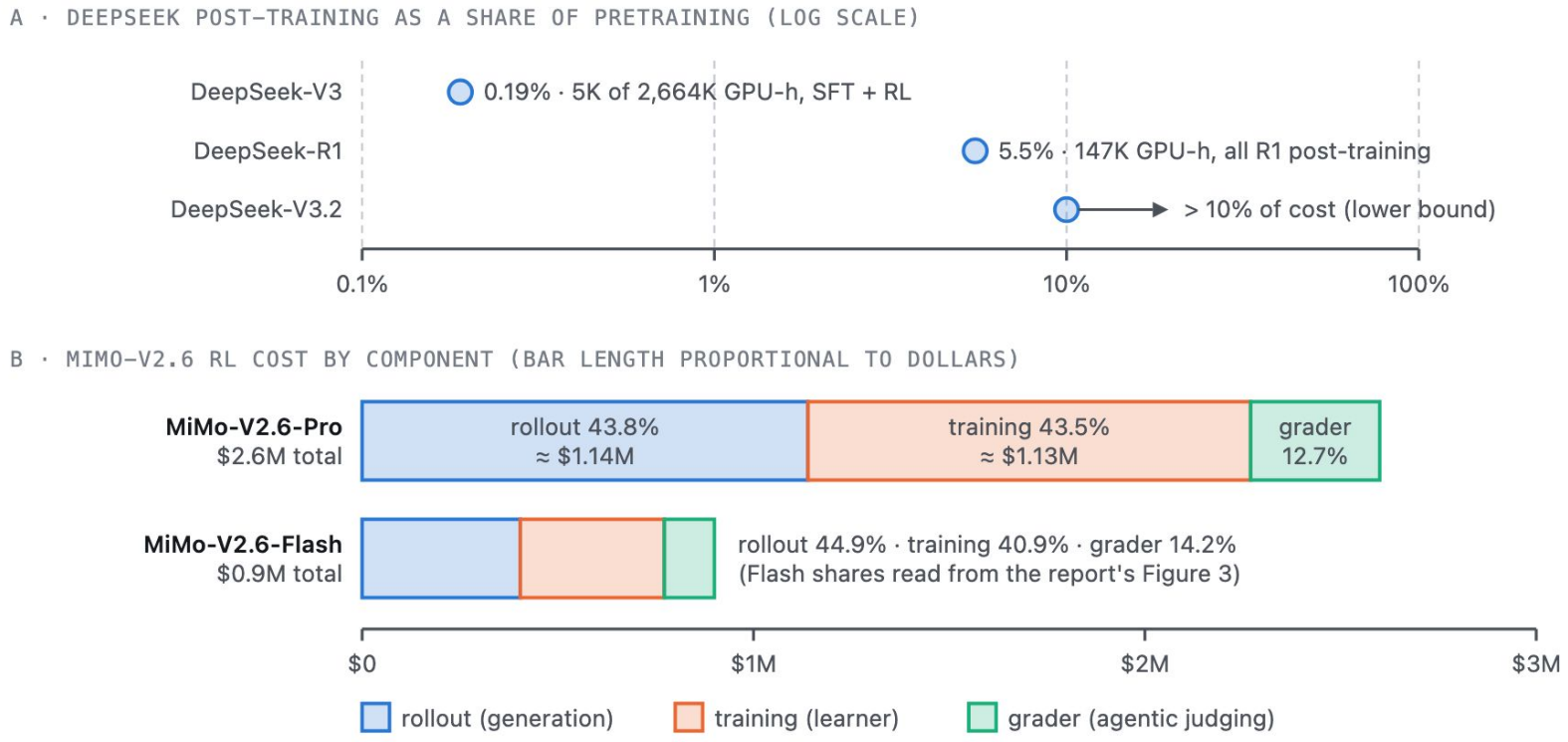
- Reasoning models think before answering; under outcome rewards alone, thinking keeps growing.
- Control moved from separate thinking models, to a thinking switch in one model, to effort levels in the prompt.

- Effort is trained by pricing length: at low effort, a long correct answer earns much less than a short one (DeepSeek-V4.1's 1–100 scale).
- Most 2026 flagships removed the off switch entirely, and effort replaced it.



Shipping: Cost and Quantization

- Labs serve the experts in 4 bits, either training through the rounding (QAT) or repairing it afterwards by distillation (NVIDIA).



Summary: Post-training at Scale

- Posttraining became a second major training stage: imitate briefly, run RL as specialists, then consolidate.
- Every RL method is one update with five choices; rewards, environments, harnesses and systems keep that update honest.
- Reading note: reports describe recipes that worked, rarely the runs that didn't.

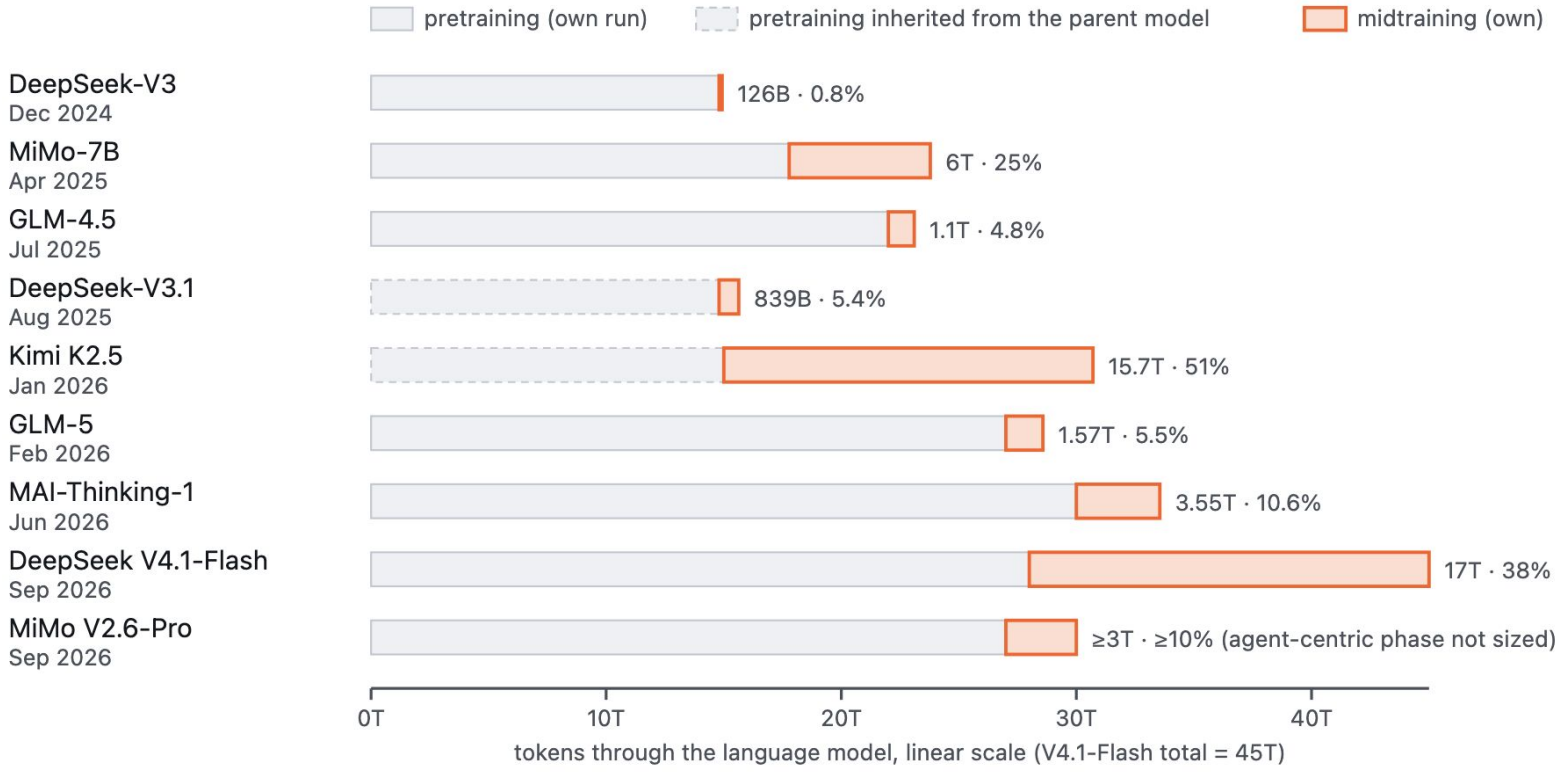
Breadth many skills in one model	**Checkable rewards** signals that resist hacking	**Rollout cost** long agent trajectories	**Stability** RL that does not collapse	**Serving and release** cheap, safe to ship
specialists per domain (07)	rule and test rewards (04)	asynchronous RL (06)	clip-higher (03)	effort levels (08)
on-policy distillation (07)	rubric reward models (04)	partial rollouts (06)	routing replay (06)	token penalties (08)
mixed-task RL (07, 12)	anti-hack guards (04)	dynamic sampling (03)	frozen router (06)	4-bit QAT (09)
synthesized environments (05)	groupwise grading (12)	fixes by distillation (11, 12)	mismatch masking (06)	safety evaluation (10)
multi-scaffold training (05)	calibration rewards (04, 10)	colocated engines (06)	critic-based PPO (03)	staged weight release (10)
			entropy control (03)	

Part 3: Mid-training



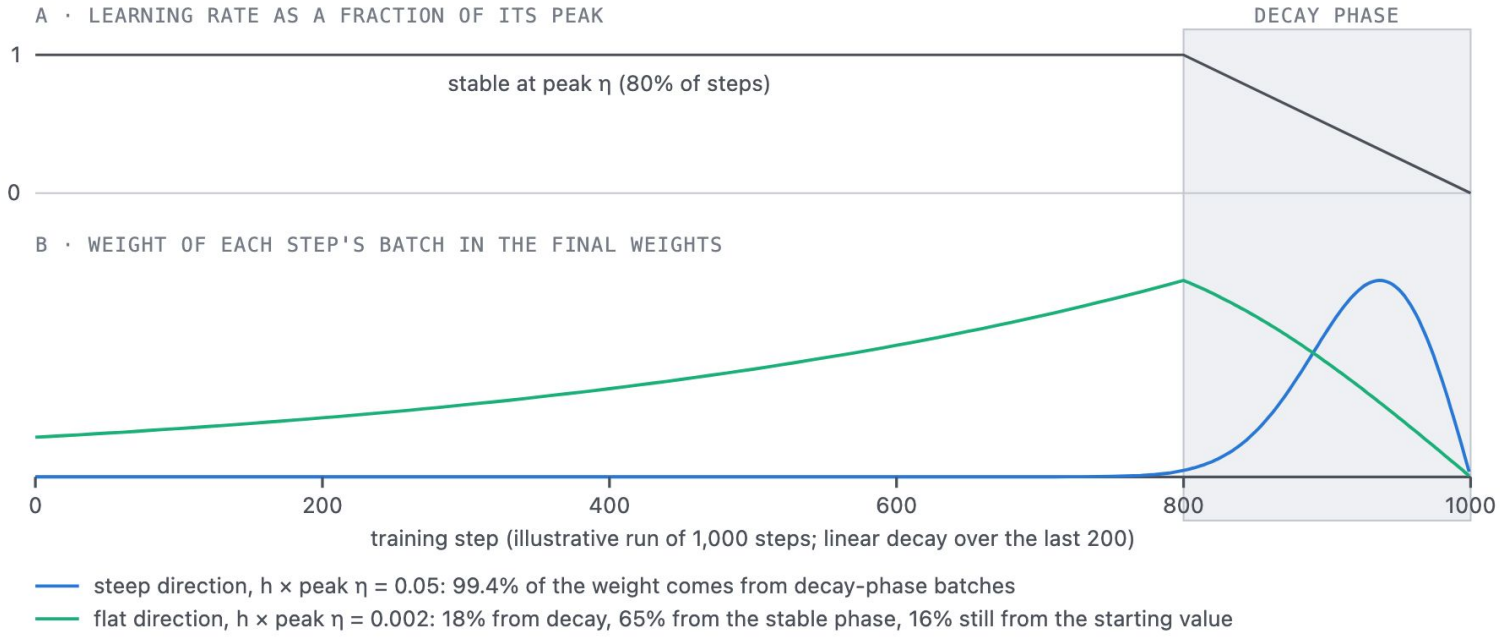
What Midtraining Is, and Why It Grew

- Midtraining continues a pretrained model with the same next-token objective but changes the data, the context length or the architecture, before SFT.

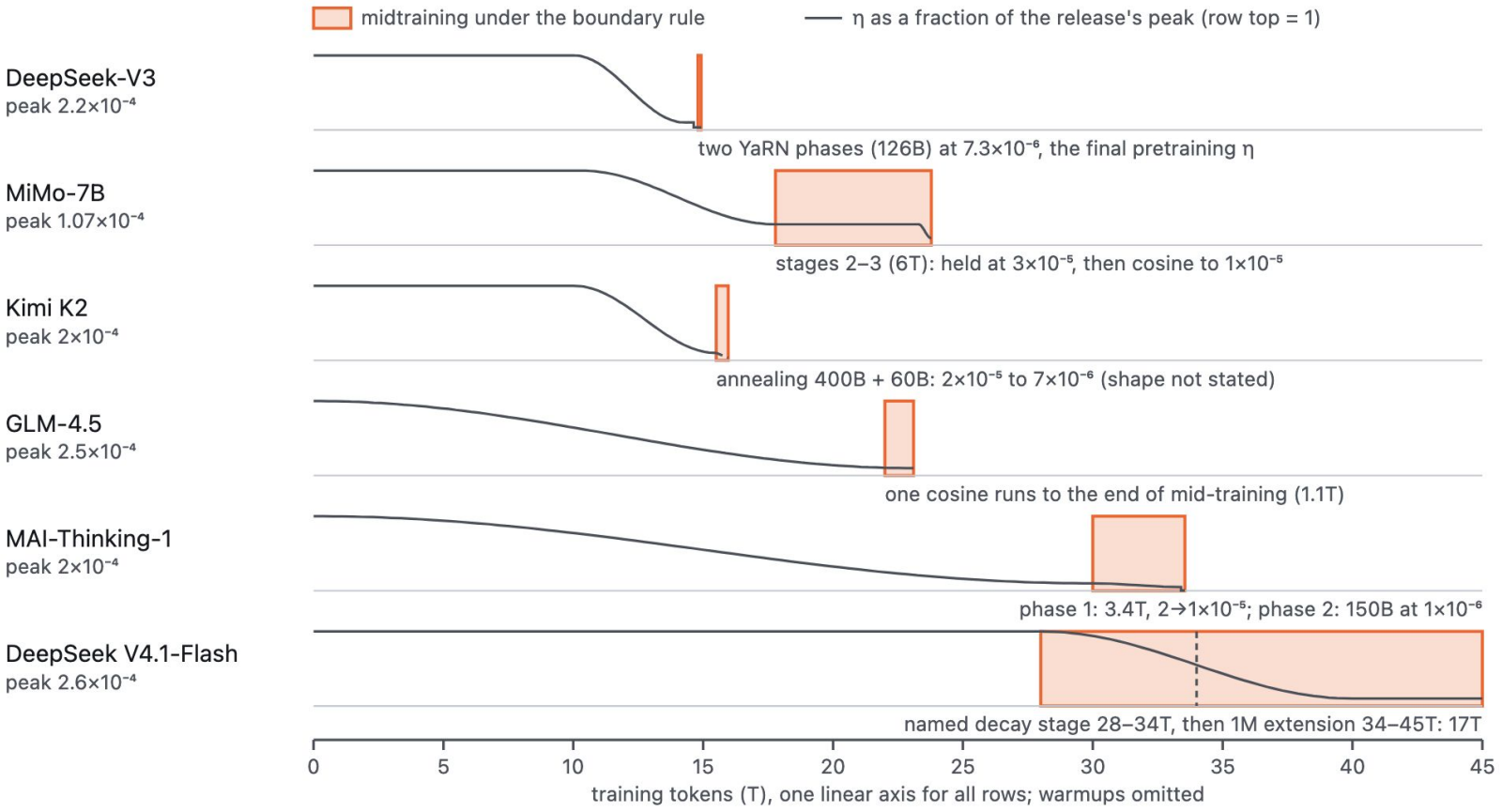


The End of the Run Weighs the Most

- Each training step pulls the weights toward its batch, and later steps partly overwrite earlier ones.
- During the lr decay, the last data sets most of the final weights in the directions where the loss is steep.

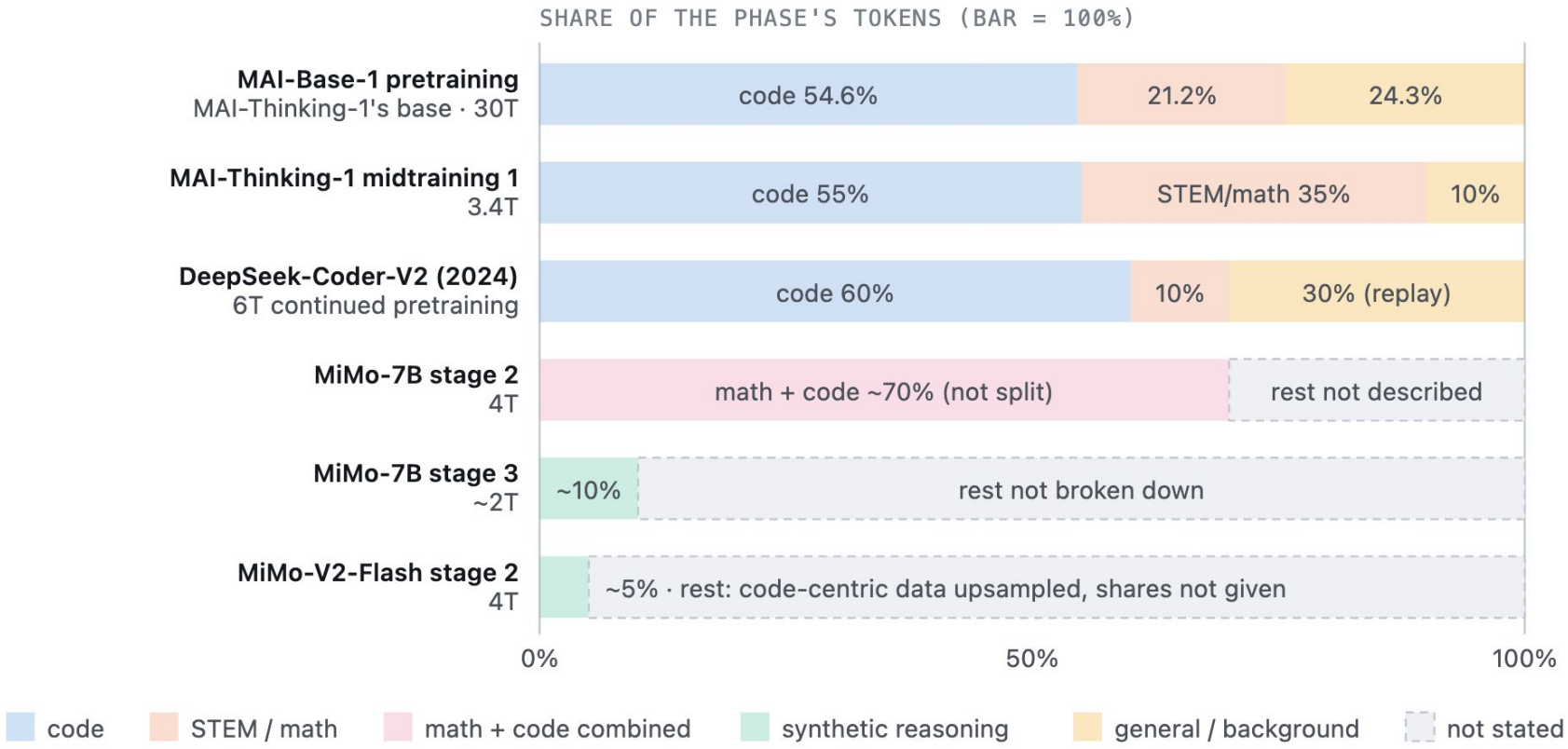


The End of the Run Weighs the Most



Choosing the Midtraining Mix

- Since 2025 the upweighted data is mostly math, code and synthetic reasoning.



Code and Agent Data

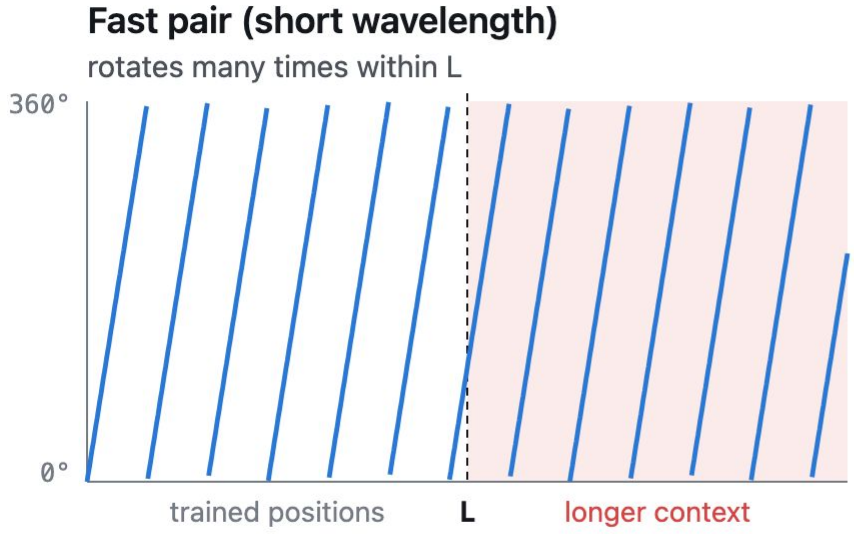
STEP	WRITTEN BY	TEXT (ILLUSTRATIVE CODING TASK)	SFT-STYLE LOSS	ALL-TOKEN LOSS
q	user	Issue: parse_date() raises AttributeError when tz=None. Fix it.	☐	■
r1	model	Find where parse_date is defined, then read it.	■	■
a1	model	`bash: grep -rn "def parse_date" src/`	■	■
o1	environment	`src/utils/dates.py:41: def parse_date(s, tz):`	☐	■
r2	model	tz.localize() runs with no None check; return d unchanged when tz is None.	■	■
a2	model	`edit src/utils/dates.py + if tz is None: return d`	■	■
o2	environment	`pytest tests/test_dates.py: 5 passed`	☐	■
r3	model	Done: parse_date now returns naive dates unchanged.	■	■

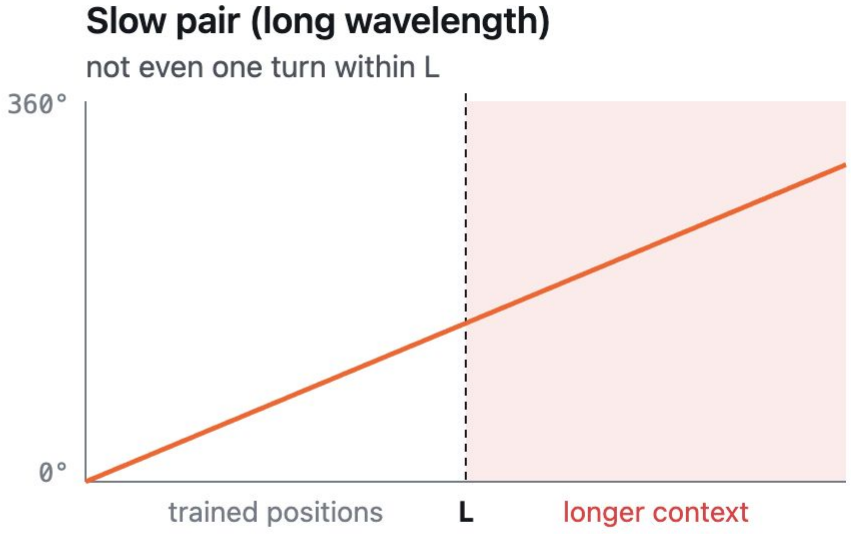
task from the user · written by the model · returned by tools · ■ token gets loss · ☐ masked, no loss

Rows follow GLM-5's notation: q, then reasoning r, action a, observation o per round.

Stretching Position Encodings

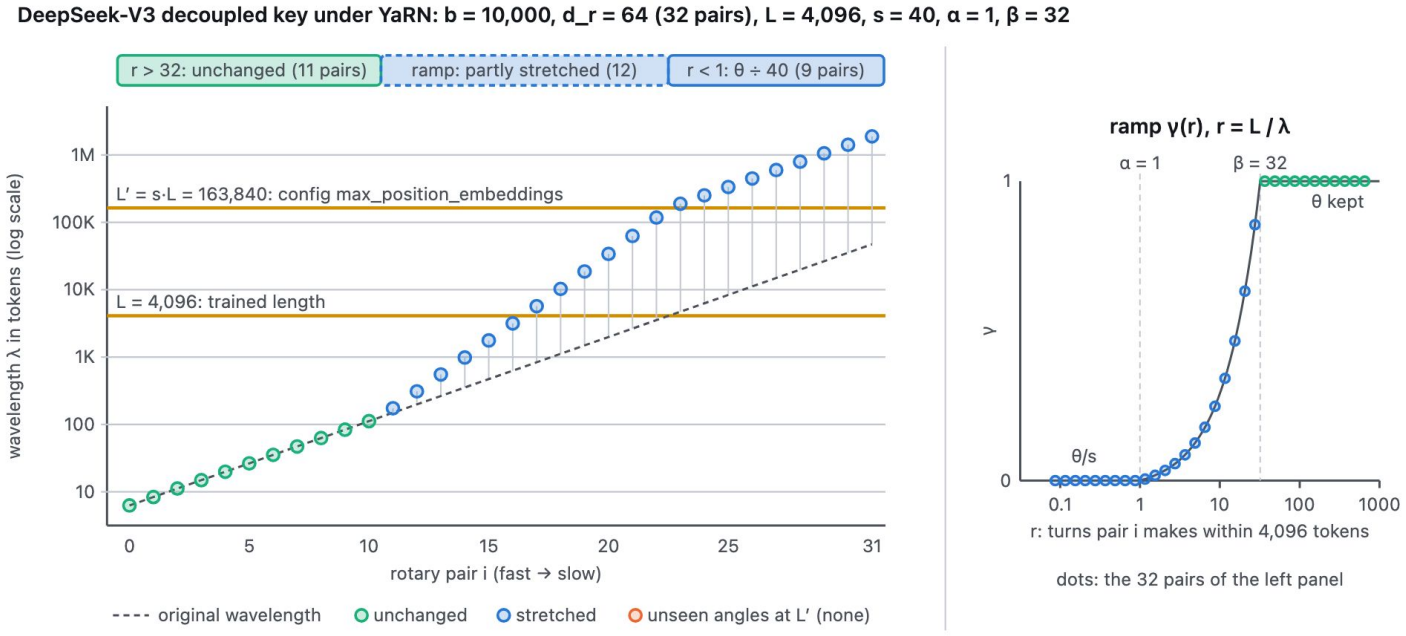
- Rotary position encoding (RoPE) rotates pairs of dimensions at different speeds; attention scores depend on the angle between query and key.
- Fast pairs turn many times within the trained length L, so every angle is familiar. Slow pairs never complete a turn, so positions past L produce angles the model never saw.



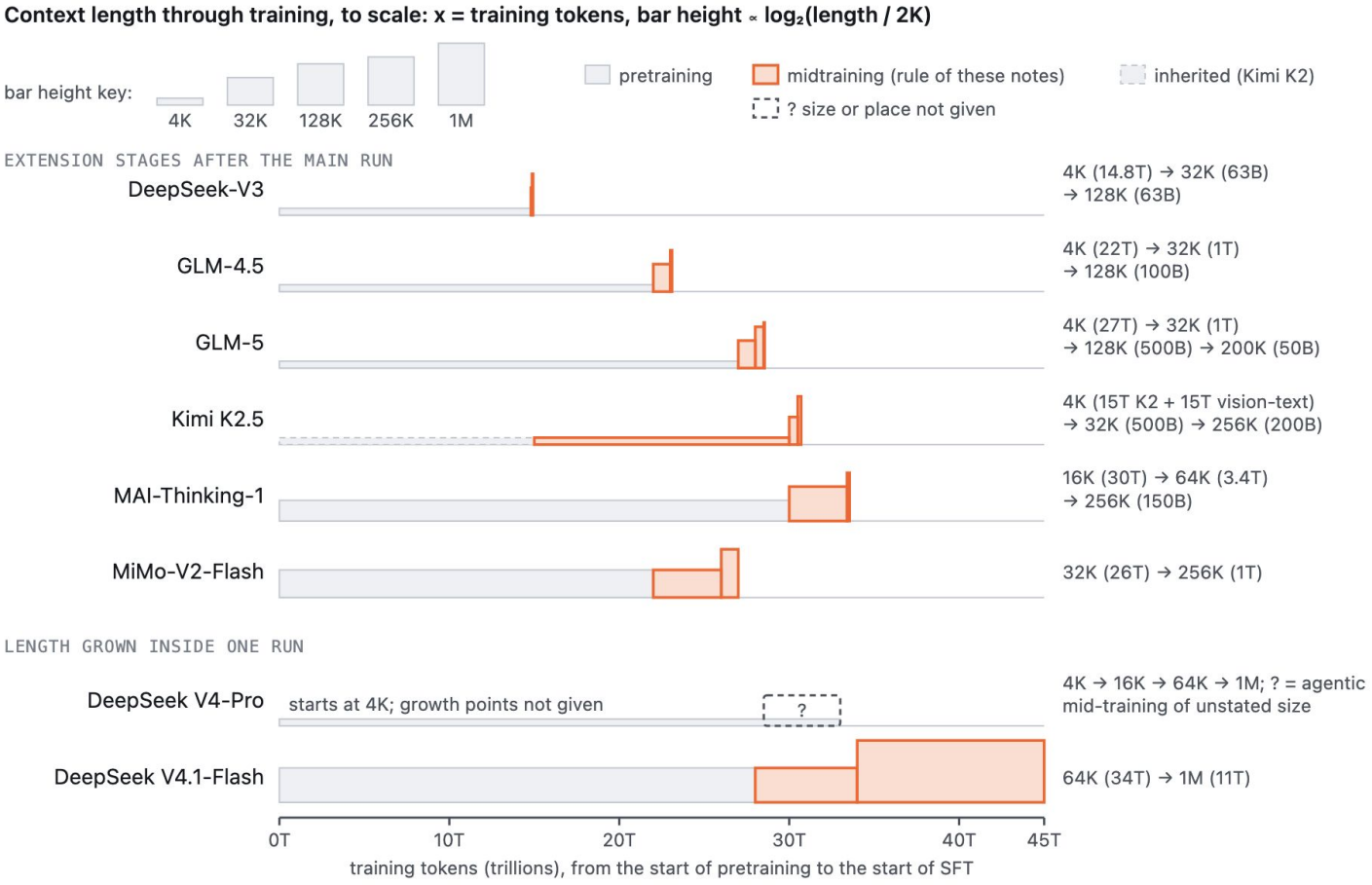


Stretching Position Encodings

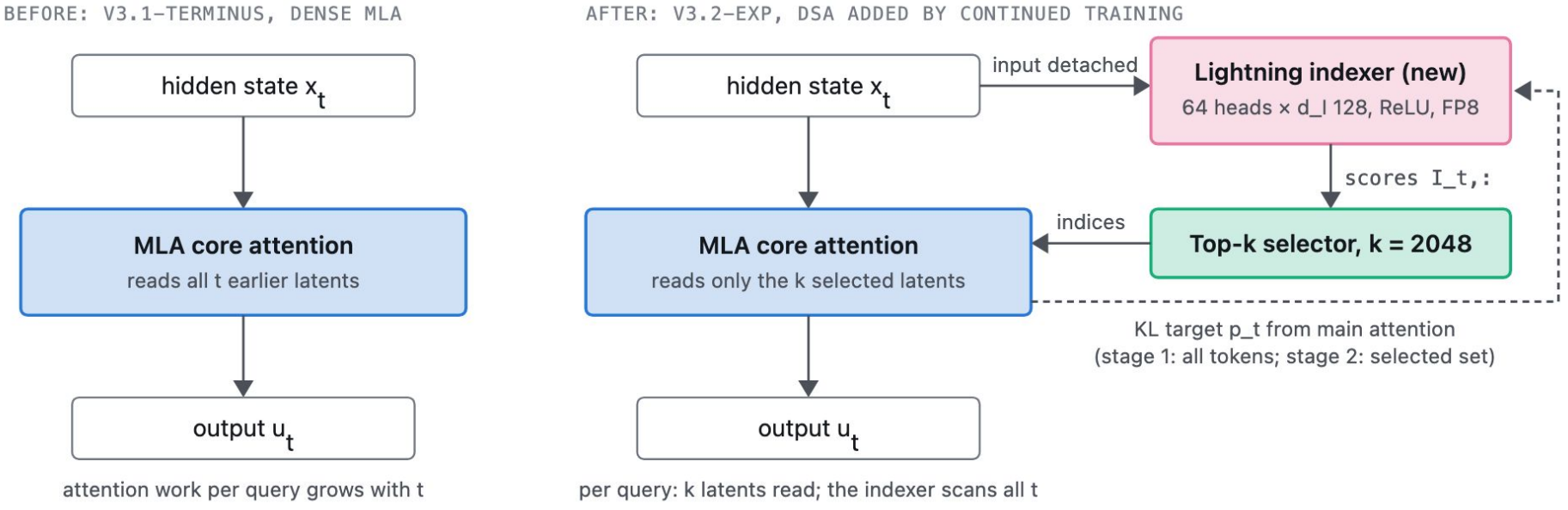
- Position interpolation, a larger rotation base and YaRN all slow the slow pairs.
- Several 2025–26 models drop position encoding from their global layers, which then need no rescaling at all.



Growing the Context in Stages



Retrofitting Architecture by Continued Training



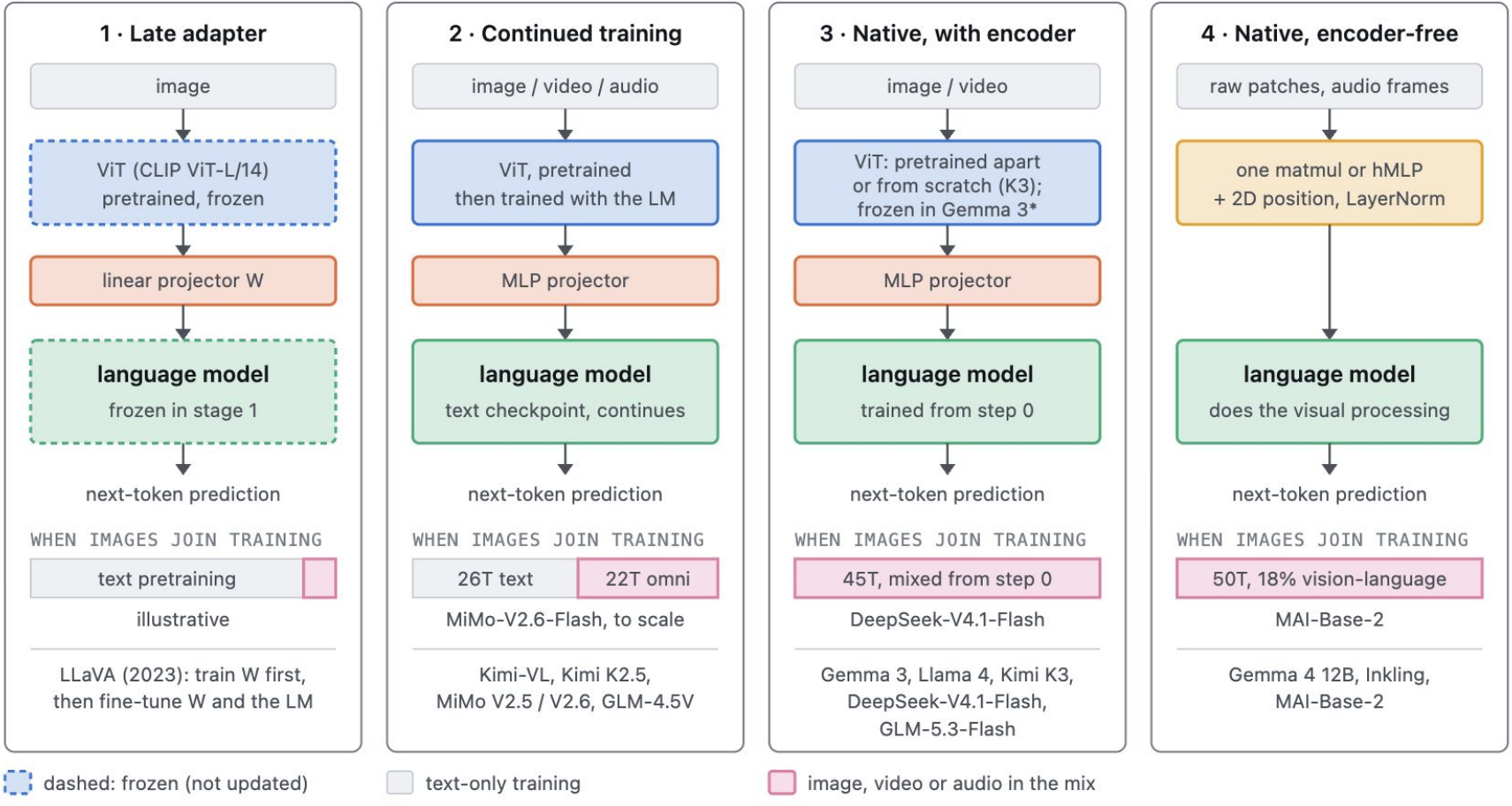
Stage 1 · dense warm-up (midtraining)

attention stays dense; all weights frozen except the indexers
indexers learn the KL loss toward dense attention
V3.2-Exp: 2.1B tokens, LR 1e-3

Stage 2 · sparse training (midtraining)

top-k on; main model trains on the LM loss only
indexers keep the KL loss on the selected set S_t
V3.2-Exp: 943.7B tokens, LR 7.3e-6, k = 2048

Multi-modalities: When and Where



Recap: What's Between Pre and Post

- Midtraining prepares a base for what comes next: long context, code and agent behaviors for RL, new modalities, and the optimizer and precision RL will use.
- It works because the end of the run weighs the most, and because a finished model can be extended instead of retrained.
- Reading note: labels differ, mixes are rarely given in numbers, and a third of releases don't describe the stage.

Use the best data last low learning rate	**Prepare for RL** skills RL can amplify	**Long context** 128K to 1M	**Change the model** without retraining	**Check and remove** what goes out
decay and annealing (01) more math and code (02) quality filtering (02) replay of general data (02)	synthetic reasoning (02) repo code, issues, PRs (03) agent trajectories (03) large-batch optimizer (08)	PI, base raise, YaRN (04) NoPE global layers (04) staged lengths, packing (05) long-range data (05)	sparse-attention retrofit (06) vision, audio encoders (07) 4-bit training starts (08) settings changed (08)	base-model evaluation (09) decontamination (09) removing risky data (09) memorization caps (02, 09)