CPEN 455: Deep Learning

Lecture 9: Large Language Models

Renjie Liao

University of British Columbia

Winter, Term 1, 2026

Outline

- Tokenization (BPE)
- Pretraining: General, Mid-Training & Context Extension
  - Next-Token Prediction & Scaling Laws
  - Mixture of Experts (MoE)
- Supervised Fine-Tuning
- Parameter-Efficient Fine-Tuning (PEFT)
  - LoRA & QLoRA
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

What Is a Token?

A token is a discrete vocabulary entry, represented by an integer ID.
A tokenizer converts text into these entries.

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!

Subwords can split a word. Here </w> marks a word ending.

Let’s </w>	do</w>	token	ization</w>	!</w>

[1] Hugging Face LLM Course.

Byte Pair Encoding (BPE)

Toy corpus: one sequence, “abcabcab”, treated as one segment.

$$V_0 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$$

Learn three merge rules. Recount adjacent pairs at each round and save the ranks.

Initial sequence	[a, b, c, a, b, c, a, b]	
Rank 1 3 pairs	[ab, c, ab, c, ab]	merge a + b
Rank 2 2 pairs	[abc, abc, ab]	merge ab + c
Rank 3 1 pair	[abcabc, ab]	merge abc + abc

For tied counts, choose the pair whose first occurrence is leftmost (toy convention).

[2] Sennrich et al., ACL 2016; [3] OpenAI, tiktoken educational implementation.

BPE Training Algorithm

Keep initial units and learned pieces in the vocabulary; learn merge ranks from counts.

Input: segmented corpus, initial alphabet, maximum number of merges.

$$C, \qquad V \leftarrow V_0, \qquad \mathcal{R} \leftarrow [\,], \qquad M \in \mathbb{N}$$

For each training iteration: $j = 1, \ldots, M$

1. Count eligible adjacent pairs. Stop if none remain; otherwise select a maximum.

$$(a, b) \leftarrow \arg\max_{(u,v)} c_C(u, v)$$

2. Replace all nonoverlapping occurrences in the current corpus, left to right.

3. Add the new piece and record this pair with the next rank.

$$V \leftarrow V \cup \{ab\}, \qquad r(a, b) \leftarrow j, \qquad \mathcal{R}.\,\mathrm{append}((a, b))$$

Return the vocabulary and ranked rules. Freeze them before encoding new text.

[2] Sennrich et al., ACL 2016; [3] OpenAI, tiktoken educational implementation.

BPE Encoding: Choosing a Merge

Encoding applies fixed learned rules. It does not retrain the tokenizer.

$\mathcal{R}$	The merge rules saved during tokenizer training.
$r(u, v)$	The saved priority of a pair. A smaller rank means higher priority.

$$S = (s_1, \ldots, s_n), \qquad E(S) = \{i : 1 \leq i < n,\ (s_i, s_{i+1}) \in \mathcal{R}\}$$

$E(S)$	Positions where two adjacent current pieces match a saved rule.

1. Stop if no pair is eligible. Otherwise choose the next merge position:

$$i^\star = \arg\min_{i \in E(S)} r(s_i, s_{i+1})$$

The argmin returns the position with the smallest rank. If tied, choose the leftmost one.

2. Merge that occurrence, update the sequence and check eligible pairs again.

3. When no rule matches, map the final pieces to their fixed vocabulary IDs.

[3] OpenAI, tiktoken educational implementation.

BPE Encoding: Following the Ranks

Apply one merge at a time to “abcab”. Numbers under pieces are current token positions.

Saved ranks: $r(\texttt{a}, \texttt{b}) = 1, \quad r(\texttt{ab}, \texttt{c}) = 2, \quad r(\texttt{abc}, \texttt{abc}) = 3$

Current pieces	Ranks at eligible positions	Decision
$[\underset{1}{\texttt{a}}, \underset{2}{\texttt{b}}, \underset{3}{\texttt{c}}, \underset{4}{\texttt{a}}, \underset{5}{\texttt{b}}]$	Position 1: rank 1 Position 4: rank 1	Same rank: choose the leftmost.
$[\underset{1}{\texttt{ab}}, \underset{2}{\texttt{c}}, \underset{3}{\texttt{a}}, \underset{4}{\texttt{b}}]$	Position 1: rank 2 Position 3: rank 1	Rank 1 beats rank 2, even farther right.
$[\underset{1}{\texttt{ab}}, \underset{2}{\texttt{c}}, \underset{3}{\texttt{ab}}]$	Position 1: rank 2	Merge ab + c.
$[\underset{1}{\texttt{abc}}, \underset{2}{\texttt{ab}}]$	None	No eligible pair: stop.

Decoding: Look up pieces by ID, then concatenate.

$$[4, 3] \mapsto [\texttt{abc}, \texttt{ab}] \mapsto \texttt{abcab}$$

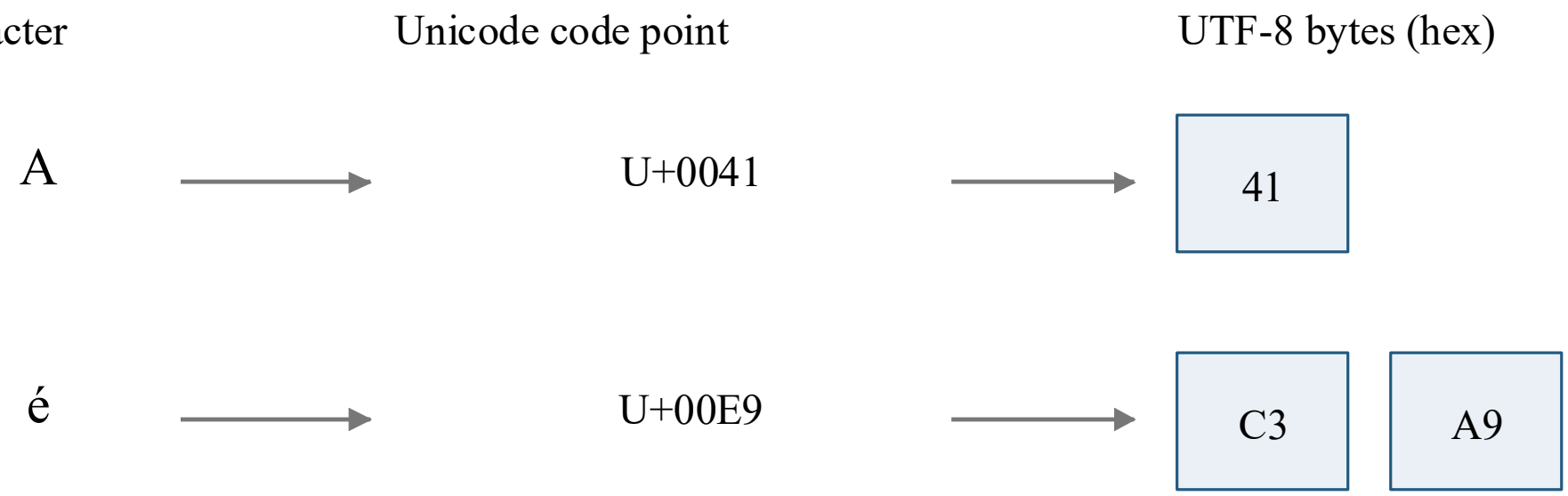
Toy IDs: a=0, b=1, c=2, ab=3, abc=4, abcabc=5.

[3] OpenAI, tiktoken educational implementation.

Unicode and UTF-8: Why They Matter

Unicode assigns standardized code points to text elements across scripts.
UTF-8 encodes these code points as bytes for storage and transmission.

Code points identify text elements. Token IDs belong to a trained tokenizer.



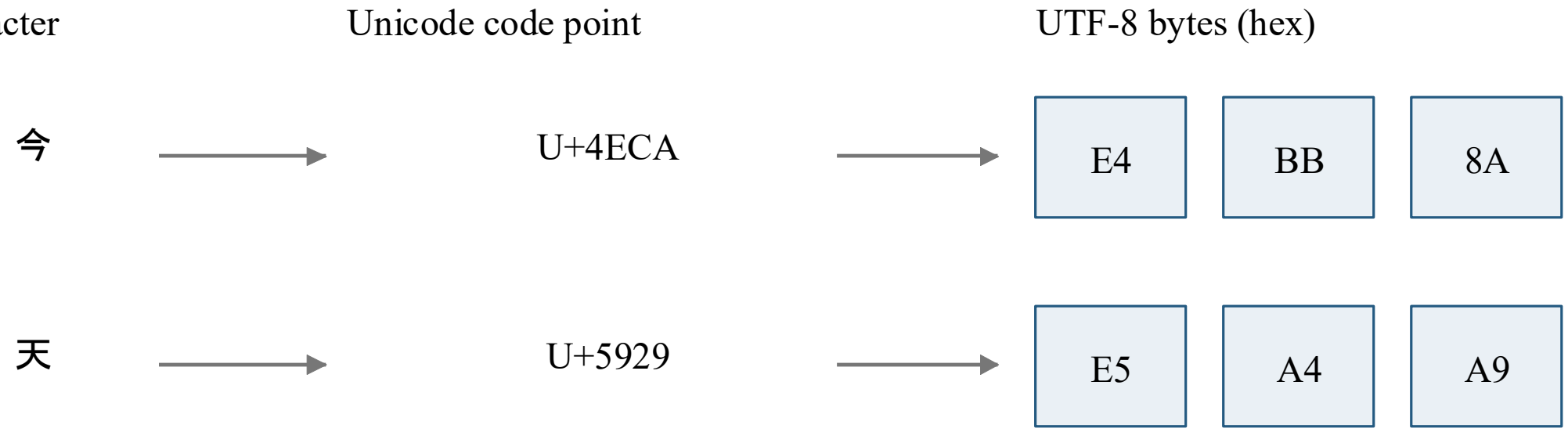
One encoded code point uses 1–4 UTF-8 bytes. Byte-level BPE merges byte sequences.
Why care? All text can start from the same alphabet of 256 byte values.

[4] Yergeau, RFC 3629, 2003, §3.

Unicode and UTF-8: An Example

Unicode identifies the characters; UTF-8 supplies bytes for byte-level BPE.

今天 “today” in Chinese

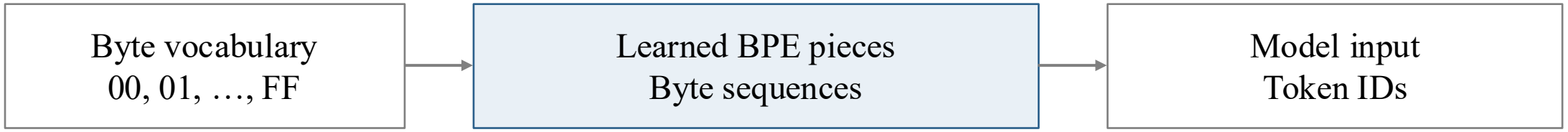


Two code points become six bytes; this does not determine the token count.

[4] Yergeau, RFC 3629, 2003, §3.

Byte-Level BPE: Coverage and Compression

UTF-8 represents text as bytes. Byte-level BPE starts from all 256 byte values.



Coverage: any valid UTF-8 text remains representable.

Compression: frequent byte sequences can use one token rather than several byte tokens.

Decoding: token IDs map back to stored byte sequences, which reconstruct the input text.

[3] OpenAI, tiktoken educational implementation; [4] Yergeau, RFC 3629, §3.

BPE in GPT Models

GPT models use byte-level BPE, but their learned vocabularies differ [5].

Model generation	tiktoken encoding	Ordinary vocabulary entries
GPT-2	gpt2 (r50k_base)	50,256
GPT-3.5-Turbo / original GPT-4	cl100k_base	100,256
GPT-4o / GPT-4.1	o200k_base	199,998

Special/control tokens are additional and model-specific.
GPT-2 adds one end-of-text token: 50,257 entries in total.

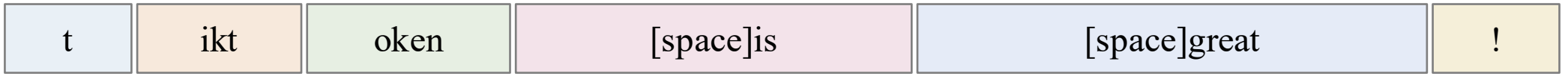
Vocabulary size is a tokenizer property; it is different from the context-window length.

[5] OpenAI, tiktoken encoding definitions and model-to-encoding mappings.

GPT Token Examples

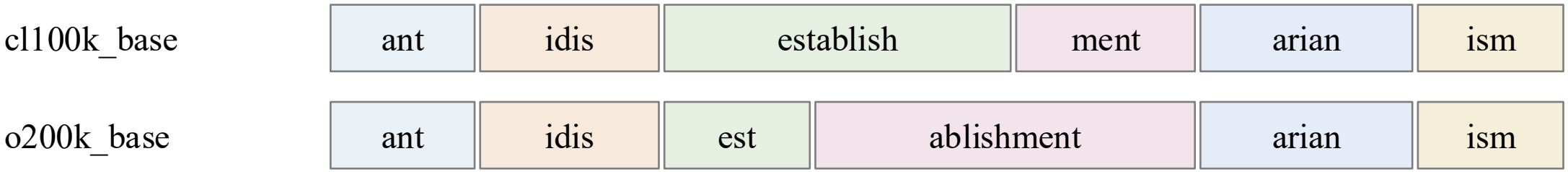
Actual token boundaries from OpenAI's published tiktoken examples [6].

GPT-4o / o200k_base: "tiktoken is great!"



Token IDs: 83, 8251, 2488, 382, 2212, 0

The same word can split differently across tokenizers:



[space] marks a space inside a token. A token is not necessarily a whole word.

[6] OpenAI Cookbook, How to count tokens with tiktoken.

WordPiece Tokenization

BERT uses a learned vocabulary of subword pieces. Encoding keeps it fixed [7].

At each position, choose the longest vocabulary prefix, consume it, and repeat.

Relevant entries: un ##aff ##able …

Step	Remaining text	Chosen piece	New suffix
1	unaffable →	un →	affable
2	affable →	##aff →	able
3	able →	##able →	empty

marks a continuation within the same word. It is absent from the input text.

If a position has no matching piece, BERT uses [UNK] for that whole word [7].

[7] Google Research, BERT WordPiece tokenizer implementation.

Unigram Tokenization

Unigram learns piece probabilities from text [8]. T5 uses this tokenizer [9].

Score a complete segmentation by multiplying the probabilities of all its pieces.

Toy vocabulary:

a	ab	bc	c
0.3	0.4	0.2	0.1

Input: abc

	Candidate pieces		**Segmentation score**
Segmentation 1	ab	c	$0.4 \times 0.1 = 0.04$
Segmentation 2	a	bc	$0.3 \times 0.2 = 0.06$

Choose [a, bc]: its complete-segmentation score is larger, even with a shorter first piece.

Dynamic programming (Viterbi) finds the best path without listing every segmentation.
The probabilities above are illustrative, not actual T5 vocabulary values.

[8] Kudo, ACL 2018, §3.2; [9] SentencePiece. Toy vocabulary and probabilities.

Tokenizer Choices Across Models

SentencePiece is a toolkit supporting BPE and Unigram [9].
Its name does not specify which segmentation algorithm a model uses.

Method	How tokens are formed	Examples
BPE	Apply learned merge ranks	GPT; Llama 2 [10]; Llama 3 [11]
WordPiece	Greedy longest vocabulary match	BERT [7]
Unigram	Highest-probability segmentation	T5 [9]
UTF-8 bytes	One byte per ordinary input token	ByT5 [12]

Llama 2 uses SentencePiece BPE; Llama 3 uses tiktoken-based BPE [10, 11].
ByT5 avoids learned subwords, usually at the cost of longer sequences [12].

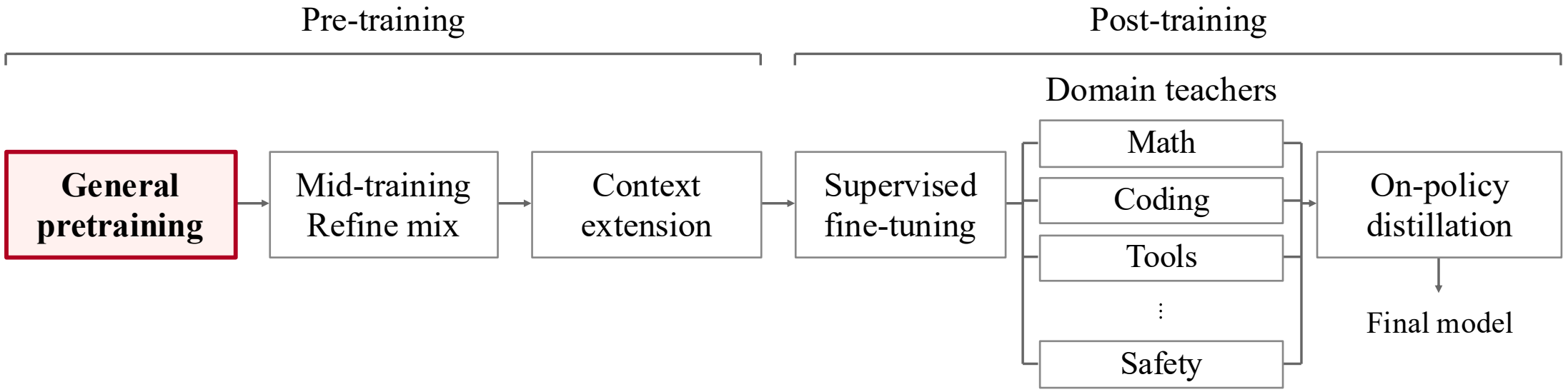
[9] SentencePiece; [10] Touvron et al., 2023; [11] Llama Team, 2024; [12] Xue et al., TACL 2022.

Outline: Pretraining

- Tokenization (BPE)
- **Pretraining: General, Mid-Training & Context Extension**
  - Next-Token Prediction & Scaling Laws
  - Mixture of Experts (MoE)
- Supervised Fine-Tuning
- Parameter-Efficient Fine-Tuning (PEFT)
  - LoRA & QLoRA
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

LLM Training Pipeline

One reported recipe: MiMo-V2-Flash [13]. Budgets and phase names follow the report.



Pretraining 1: General Pretraining

Data: 22 trillion tokens (0–22T cumulative); train on sequences up to 32K tokens.

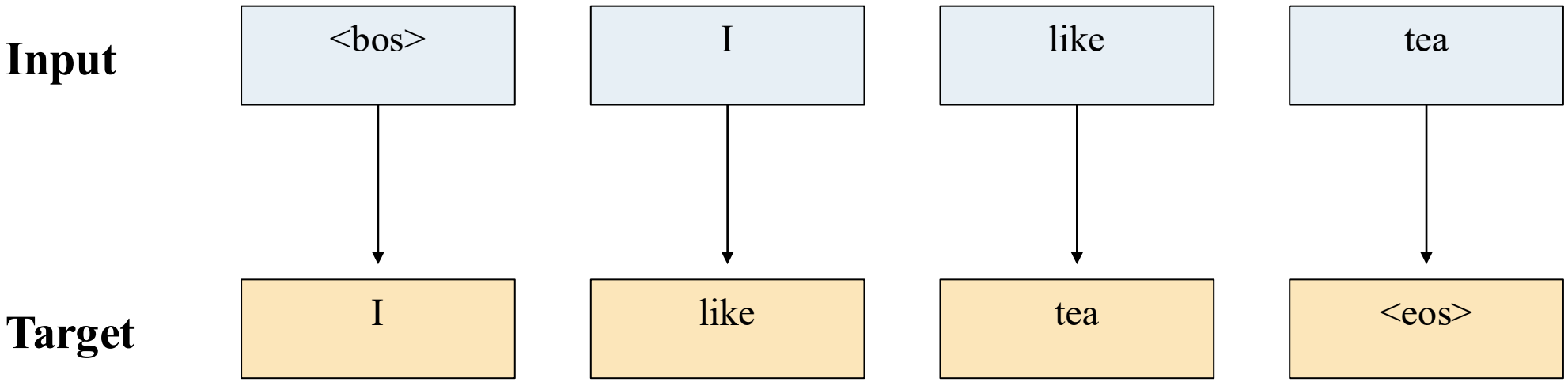
Signal: predict tokens in a broad corpus of web text, books, papers, code and STEM.

Purpose: learn broad language, knowledge and initial problem-solving capabilities.

[13] MiMo Team, MiMo-V2-Flash Technical Report, 2026, §§3.1–3.2.

Training Inputs and Targets

Shift the sequence by one position. Predict the next token at each position.

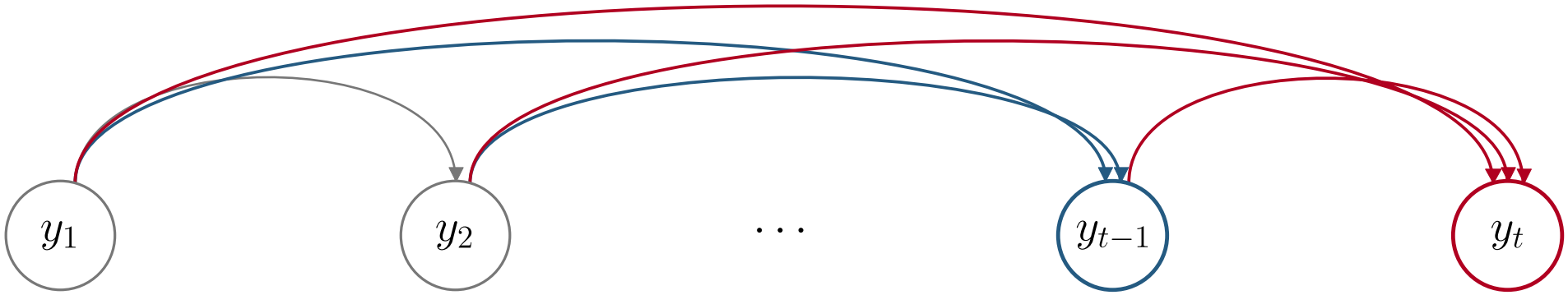


Known training prefixes allow parallel computation. Causality still applies.

Autoregressive Language Modeling

A decoder-only Transformer predicts each token from its preceding prefix.

$$\pi_\theta(y) = \prod_{t=1}^{T} \pi_\theta(y_t \mid y_{<t})$$



Arrows indicate conditioning; intermediate tokens and edges are omitted.

π_θ is parameterized by a decoder-only Transformer with learned parameters.

Learning the Next-Token Distribution

A training example is a token sequence, including its end-of-sequence marker.

$$\mathcal{L}_{\mathrm{LM}} = -\frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta(y_t \mid y_{<t})$$

Average negative log-likelihood (NLL), or average next-token cross-entropy.

Observed token	Model probability	Contribution to loss
tea	0.8	$-\log(0.8) \approx 0.223$

Assigning more probability to the observed next token lowers the loss.

$$H(q_t, \pi_\theta) = -\sum_{v \in V} q_t(v) \log \pi_\theta(v \mid y_{<t}) = -\log \pi_\theta(y_t \mid y_{<t})$$

One-hot target: the observed token. Classification classes: all vocabulary entries.

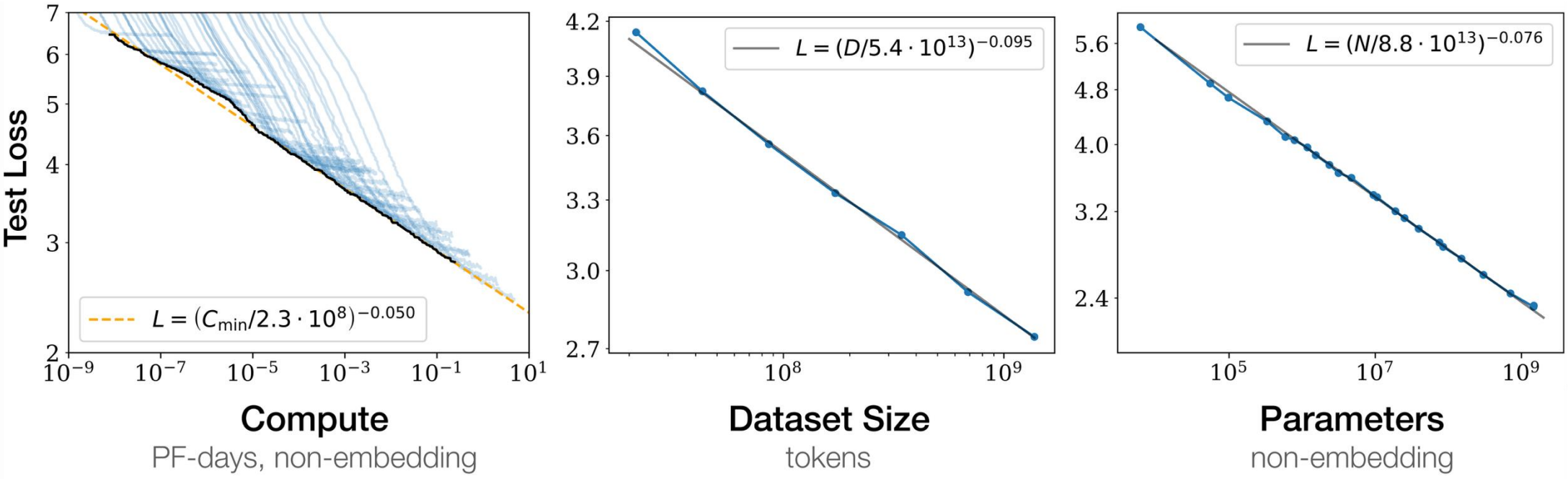
[14] Brown et al., NeurIPS 2020.

Outline: Scaling Laws

- Tokenization (BPE)
- Pretraining: General, Mid-Training & Context Extension
  - Next-Token Prediction & **Scaling Laws**
  - Mixture of Experts (MoE)
- Supervised Fine-Tuning
- Parameter-Efficient Fine-Tuning (PEFT)
  - LoRA & QLoRA
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

Pretraining Scaling Laws

Empirical fits for autoregressive pretraining on a fixed data recipe.



Next-token loss falls with scale. Many downstream capabilities improve too [14].

Changing the mixture or context length requires new measurements.

[15] Kaplan et al., 2020, Fig. 1 and §2.

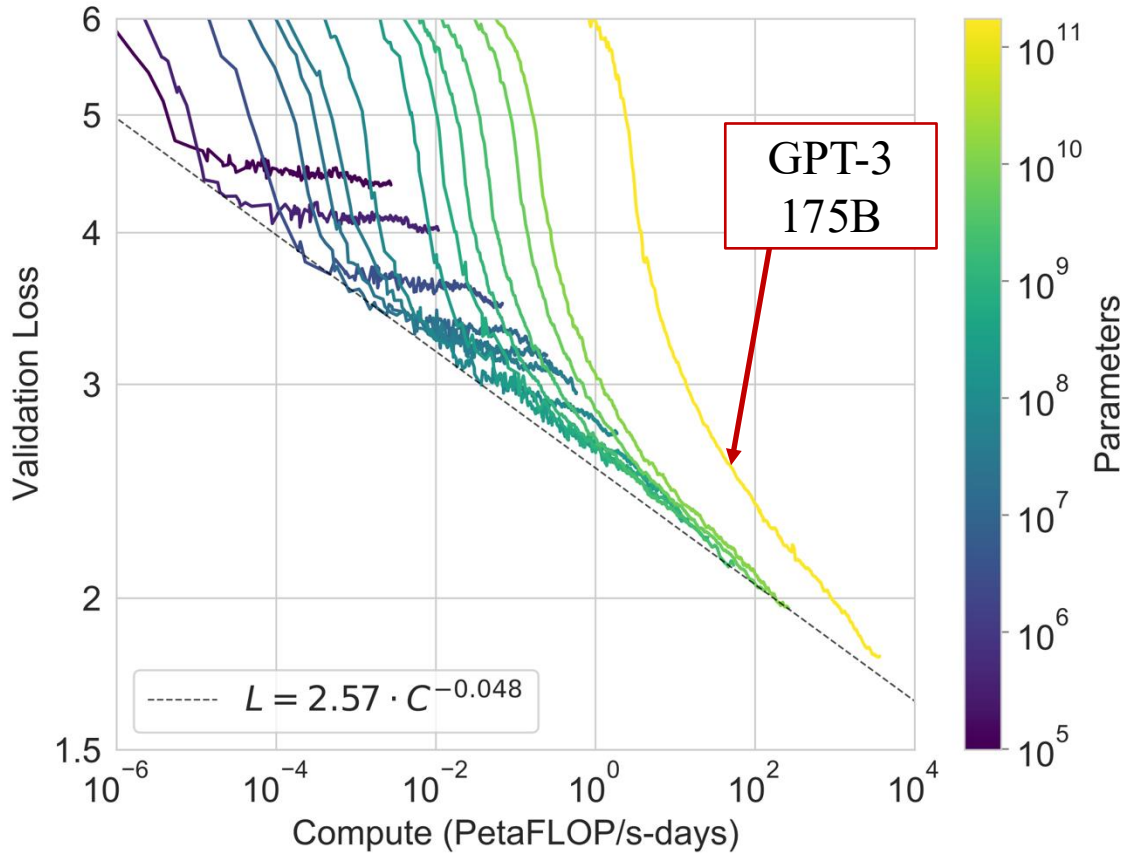
Compute-Efficient Training

A training run trades model size against training duration.

1. Each colored curve trains one fixed model size for longer.

2. More steps eventually yield diminishing improvements for that size.

Choose model size and training duration for the compute budget.



[14] Brown et al., NeurIPS 2020, Fig. 3.1.

Scaling Model Size and Training Data

At a fixed training budget, a larger model leaves fewer tokens to train on.

$$\mathcal{L}(N, D) \approx E + AN^{-\alpha} + BD^{-\beta}$$

$$C \approx 6ND$$

N: model parameters

More capacity to fit patterns.

D: training tokens

More examples and optimization.

L: held-out loss. C: training FLOPs. E, A, B, α, β: positive fitted constants.

[16] Hoffmann et al., Chinchilla, NeurIPS 2022.

The Same Compute Can Train Different Models

Illustrative dense-model allocations with the same product N × D.

Parameters N	Tokens D	Approximate C
0.5 billion	40 billion	1.2×10^{20} FLOPs
1 billion	20 billion	1.2×10^{20} FLOPs
2 billion	10 billion	1.2×10^{20} FLOPs

More parameters alone do not determine which allocation learns best.

[16] Hoffmann et al., NeurIPS 2022.

Why Next-Token Prediction Is Powerful

Text provides its own targets. Many tasks can be written as text continuation.
Better prediction can benefit from learning facts, task structure and computational patterns.

Task	Prefix supplied as context	Continuation to predict
Knowledge	The capital of France is	Paris
Translation	English: cat. French:	chat
Computation	Q: 7 + 5? A:	12

Illustrative continuations, not measured model outputs.

Under the same objective, scaling improves translation, QA and in-context learning [14].
Some benchmark capabilities become apparent only in larger models [17].
Some metrics produce abrupt score jumps even when predictions improve gradually [18].

[14] Brown et al., NeurIPS 2020; [17] Wei et al., TMLR 2022; [18] Schaeffer et al., NeurIPS 2023.

Outline: Mixture of Experts

- Tokenization (BPE)
- Pretraining: General, Mid-Training & Context Extension
  - Next-Token Prediction & Scaling Laws
  - **Mixture of Experts (MoE)**
- Supervised Fine-Tuning
- Parameter-Efficient Fine-Tuning (PEFT)
  - LoRA & QLoRA
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

Why Mixture of Experts?

Larger dense FFNs offer more parameter capacity, but use all FFN weights for every token.

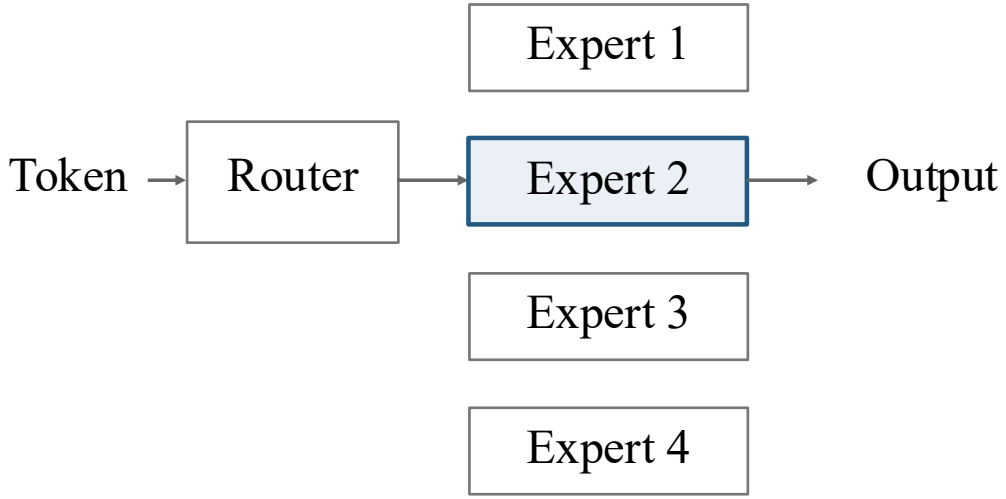
One unit means one expert's FFN weights. Counts omit the router and shared layers.

Larger dense FFN



Stored FFN weights: 4 units
Used per token: 4 units

MoE: four experts, top-1



Stored FFN weights: 4 units
Used per token: 1 unit

MoE can increase stored FFN capacity without executing every expert for every token [19].

Costs: all experts need memory; routing can add communication and uneven expert loads.

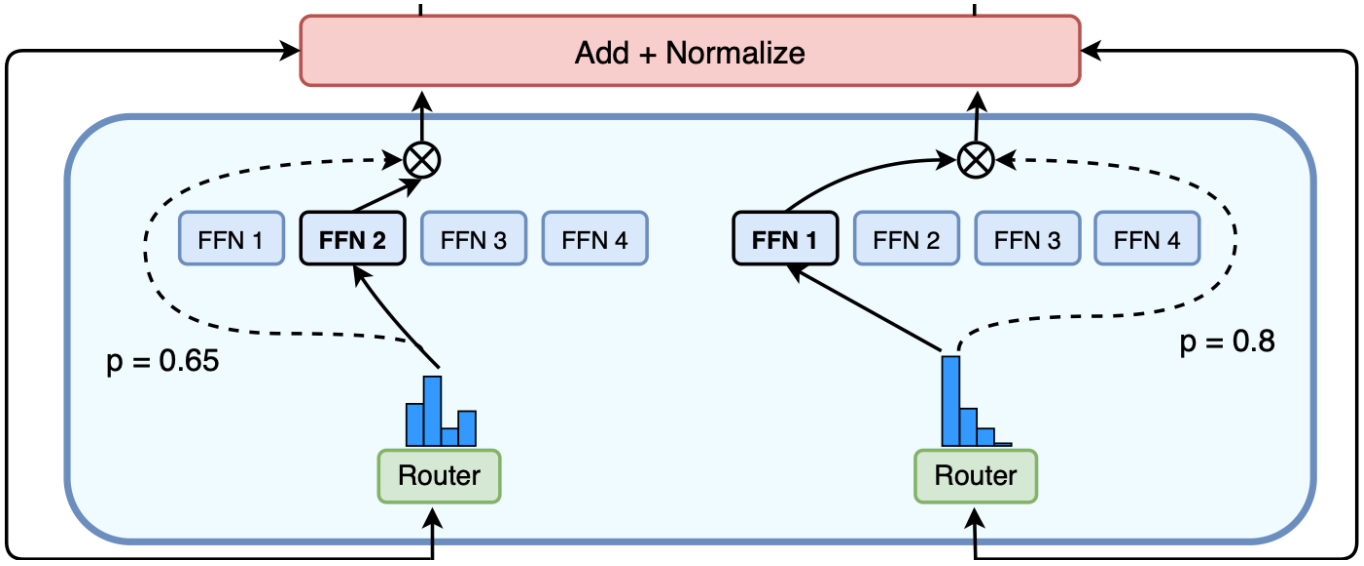
[19] Fedus et al., Switch Transformers, JMLR 2022, §2.1. Toy parameter-count illustration.

Mixture of Experts (MoE): Gating

In an MoE model, pretraining learns the router and expert FFNs jointly [19].
Softmax ranks the experts; top-k selects which expert FFNs run.

$$\mathcal{T}_k(v) = \mathrm{TopK}(\mathrm{softmax}(W_{\mathrm{r}} v),\ k)$$

$$z(v) = \sum_{e \in \mathcal{T}_k(v)} \left[\mathrm{softmax}(W_{\mathrm{r}} v)\right]_e f_e(v)$$



W_{r} learned router matrix

$\mathcal{T}_k(v)$ selected expert indices

$f_e(v)$ expert FFN output

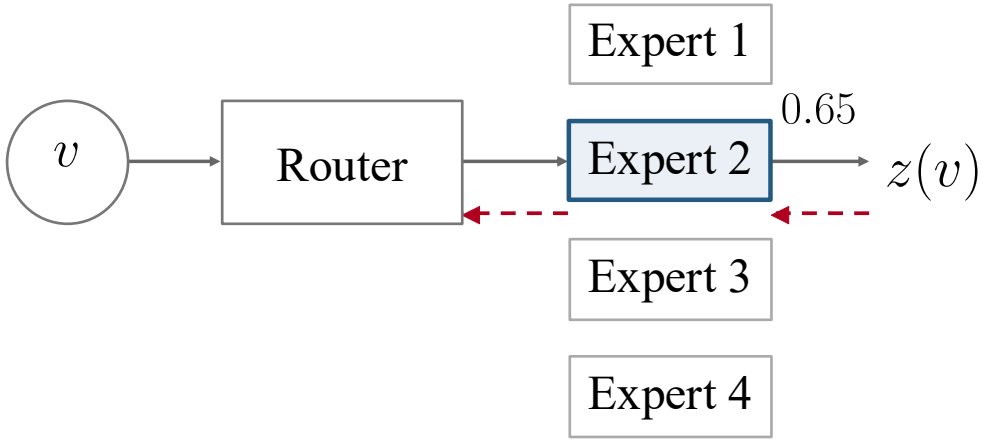
[19] Fedus et al., Switch Transformers, JMLR 2022, §2.1, Eqs. (1)–(2), Fig. 2.

MoE: Training and Inference

Switch-style top-1: hard selection with continuous gate weights in both phases.

$$p(v) = \mathrm{softmax}(W_r v) = (0.10, 0.65, 0.15, 0.10)$$

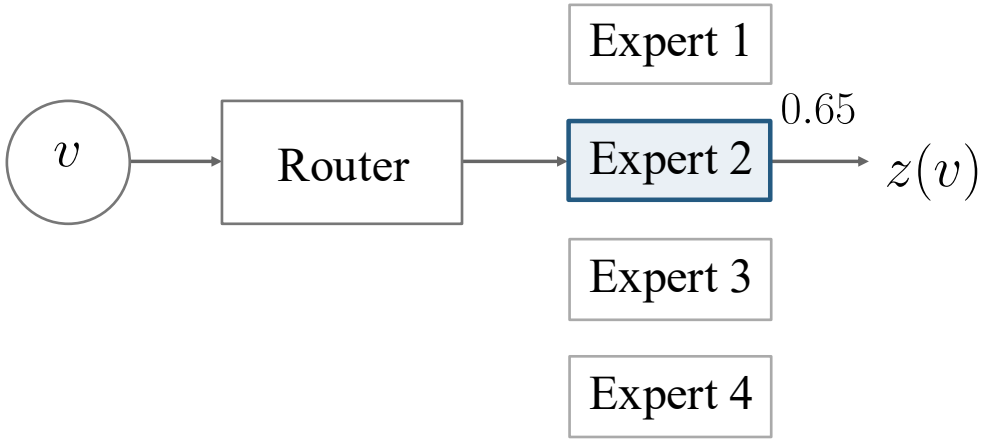
Training



task gradients

Update the router and selected expert.
Differentiate through gate values.

Inference



fixed parameters

Use the same hard top-1 choice.
Keep learned parameters fixed.

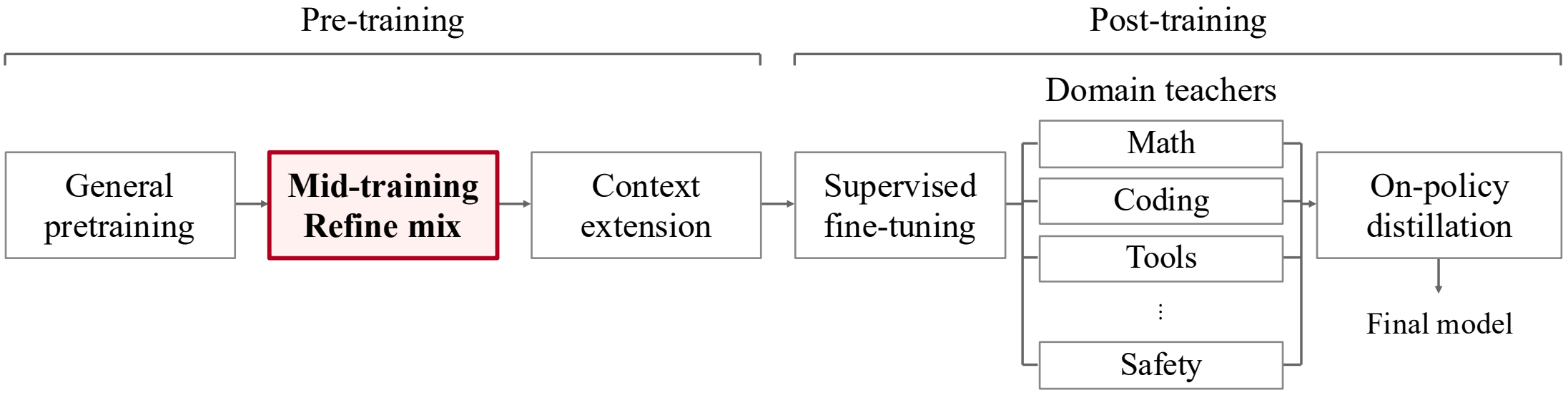
Same weighted output in both phases:

$$z(v) = 0.65 f_2(v)$$

[19] Fedus et al., Switch Transformers, JMLR 2022, §2.1, Eqs. (1)–(2).

LLM Training Pipeline

One reported recipe: MiMo-V2-Flash [13]. Budgets and phase names follow the report.



Pretraining 2: Mid-Training

Data: 4T more tokens (22–26T cumulative), with more code and about 5% synthetic reasoning.

Signal: continue token prediction while changing which sources appear more often.

Purpose: strengthen reasoning and programming before extending the context.

[13] MiMo Team, MiMo-V2-Flash Technical Report, 2026, §3.1.

What Is a Data Mixture?

A data mixture specifies the share of training tokens sampled from each source.

$$\mathcal{L}(\theta) = \sum_{k=1}^{K} \alpha_k \, \mathbb{E}_{(y_{<t}, y_t) \sim \mathcal{D}_k} [-\log \pi_\theta(y_t \mid y_{<t})]$$

K source count α_k token share $\mathcal{D}_k$ source distribution

$$\alpha_k \geq 0, \qquad \sum_{k=1}^{K} \alpha_k = 1$$

Upsampling increases a source's token share.
Its prediction targets then contribute more to the loss.

The mixture changes the emphasis while keeping the same next-token prediction objective.

[13] MiMo Team, 2026, §3.1; [11] Llama Team, 2024, §3.1.2.

Sources in a Training Mixture

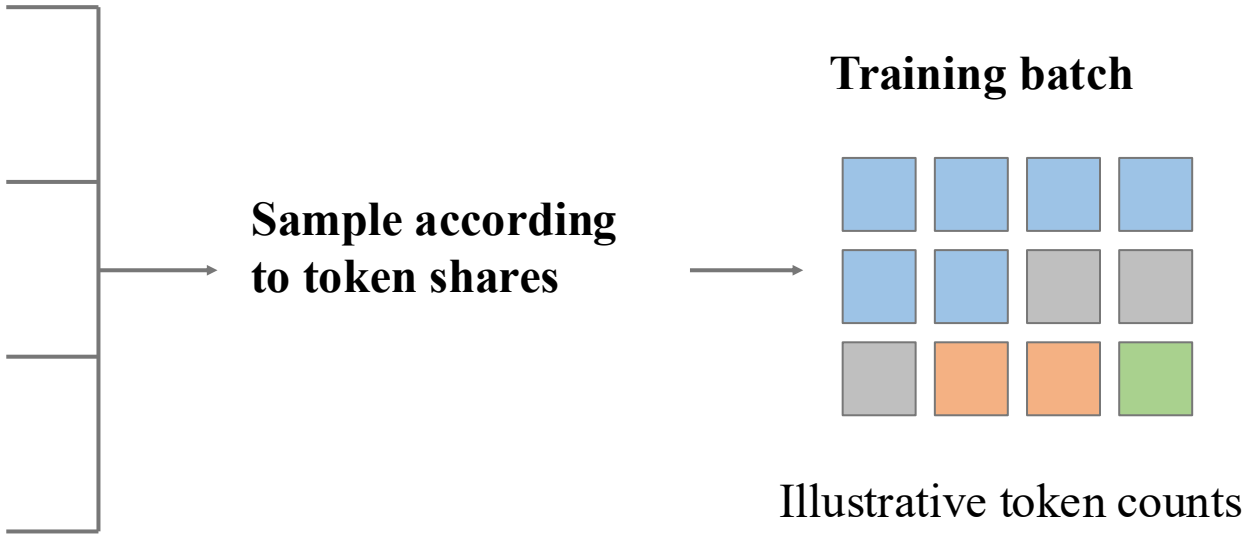
Different sources provide different next-token prediction targets.

Web and reference text
Web pages, Wikipedia and books

Code
Repositories and documentation

Math and reasoning
Worked problems and synthetic solutions

Multilingual text
Text from different languages



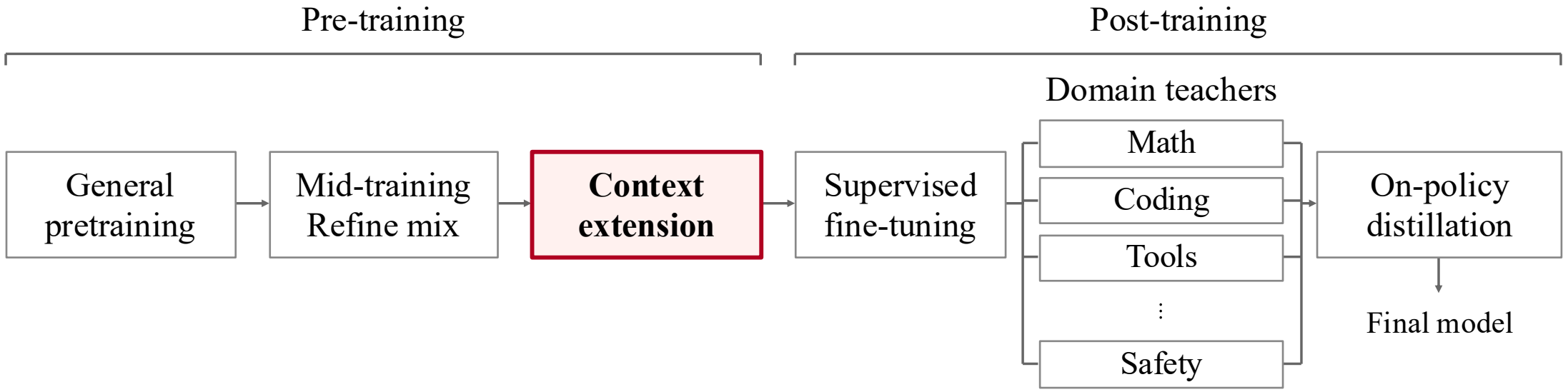
Llama 3 mixes general knowledge, math/reasoning, code and multilingual tokens [11].

MiMo increases code and uses about 5% synthetic reasoning in mid-training [13].

[13] MiMo Team, 2026, §3.1; [14] Brown et al., 2020, §2.2; [11] Llama Team, 2024, §3.1.2.

LLM Training Pipeline

One reported recipe: MiMo-V2-Flash [13]. Budgets and phase names follow the report.



Pretraining 3: Context Extension

Data: 1T more tokens (26–27T cumulative), emphasizing long-range dependencies.

Training sequences: increase from 32K to 256K tokens and adjust RoPE settings.

Purpose: learn to use evidence across long documents and code repositories.

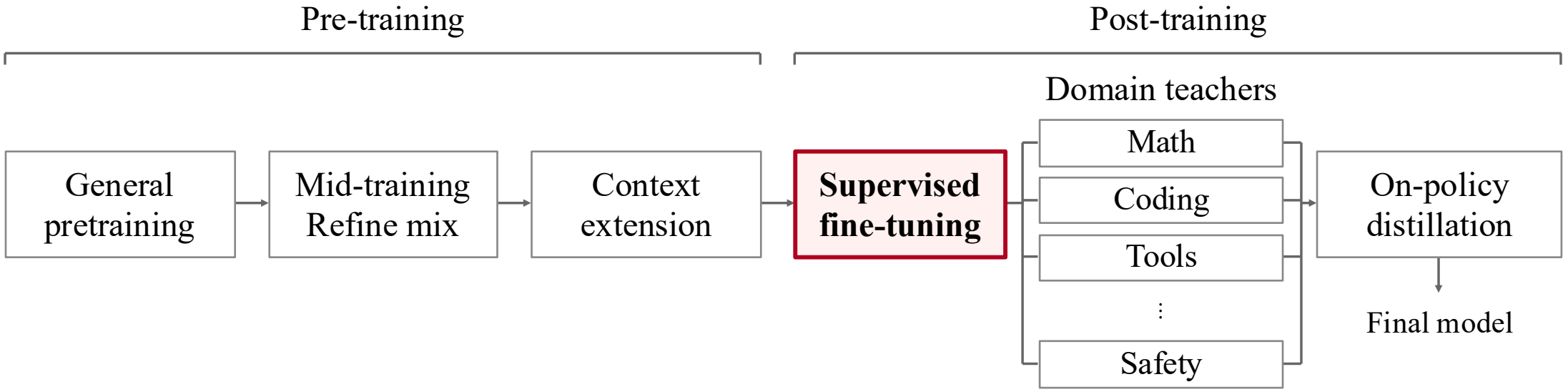
[13] MiMo Team, MiMo-V2-Flash Technical Report, 2026, §§3.1–3.2.

Outline: Supervised Fine-Tuning

- Tokenization (BPE)
- Pretraining: General, Mid-Training & Context Extension
  - Next-Token Prediction & Scaling Laws
  - Mixture of Experts (MoE)
- **Supervised Fine-Tuning**
- Parameter-Efficient Fine-Tuning (PEFT)
  - LoRA & QLoRA
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

LLM Training Pipeline

One reported recipe: MiMo-V2-Flash [13]. Budgets and phase names follow the report.



Post-Training 1: Supervised Fine-Tuning

Data: millions of instruction–response demonstrations across conversation, code and reasoning.

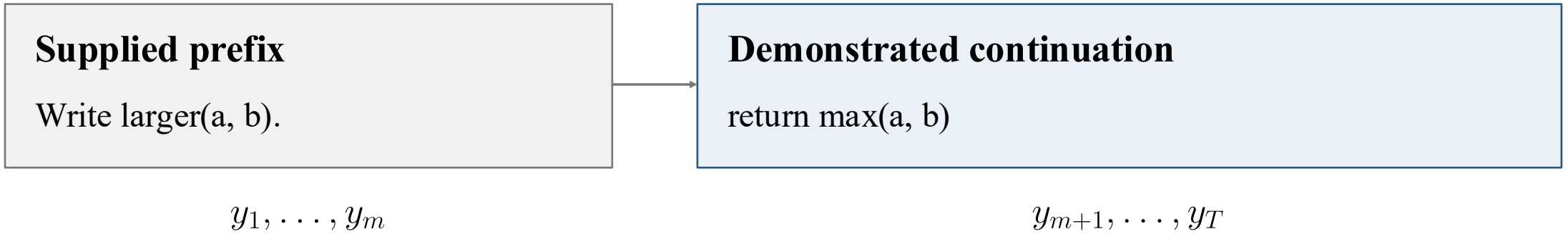
Signal: learn demonstrated responses, including thinking, non-thinking and agent formats.

Purpose: turn the base model into an instruction-following assistant.

[13] MiMo Team, MiMo-V2-Flash Technical Report, 2026, §§4.1–4.2.

Supervised Fine-Tuning

Use demonstrations of desired assistant responses.



$$\mathcal{L}_{\mathrm{SFT}} = -\mathbb{E}_{(y,m)\sim\mathcal{D}_{\mathrm{demo}}} \sum_{t=m+1}^{T} \log \pi_\theta(y_t \mid y_{<t})$$

Train on response tokens. The prefix supplies context.

[20] Ouyang et al., NeurIPS 2022.

Which Tokens Receive the SFT Loss?

The supplied prompt remains context. Only demonstrated response positions are targets.

Prompt tokens Positions 1, …, m	Response tokens Positions m + 1, …, T
Loss mask: 0	**Loss mask: 1**

$$m_t = \begin{cases} 0, & t \le m, \\ 1, & t > m, \end{cases}$$

$$\mathcal{L}_{\mathrm{SFT}}(y, m) = -\sum_{t=1}^{T} m_t \log \pi_\theta(y_t \mid y_{<t})$$

Loss masking chooses targets. The causal attention mask chooses visible context.

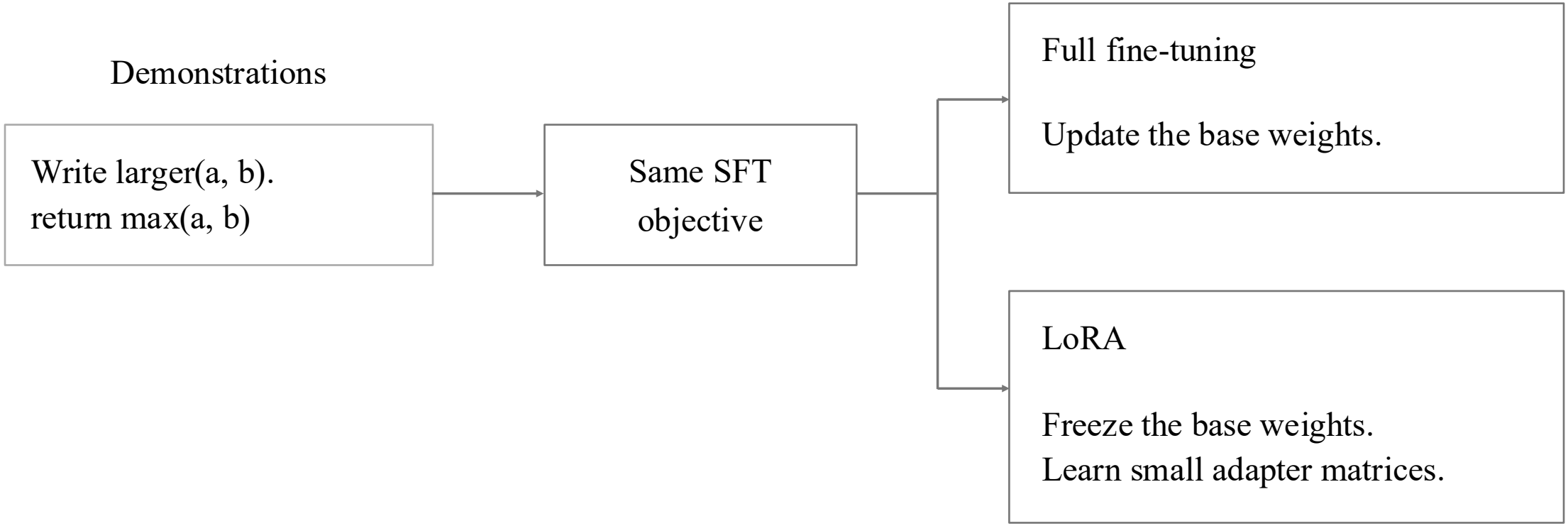
[20] Ouyang et al., NeurIPS 2022.

Outline: Parameter-Efficient Fine-Tuning

- Tokenization (BPE)
- Pretraining: General, Mid-Training & Context Extension
  - Next-Token Prediction & Scaling Laws
  - Mixture of Experts (MoE)
- Supervised Fine-Tuning
- **Parameter-Efficient Fine-Tuning (PEFT)**
  - **LoRA & QLoRA**
- Domain-Specialized Teachers
  - RL Foundations (MDPs, Returns & Policy Gradients)
  - Reward Models, PPO/GRPO & DPO
- Multi-Teacher On-Policy Distillation
- Final Model
  - Generation & Efficient Inference
  - Reasoning, Tools & Evaluation

Parameter-Efficient Fine-Tuning (PEFT)

SFT defines the learning objective. PEFT limits which parameters are trained.

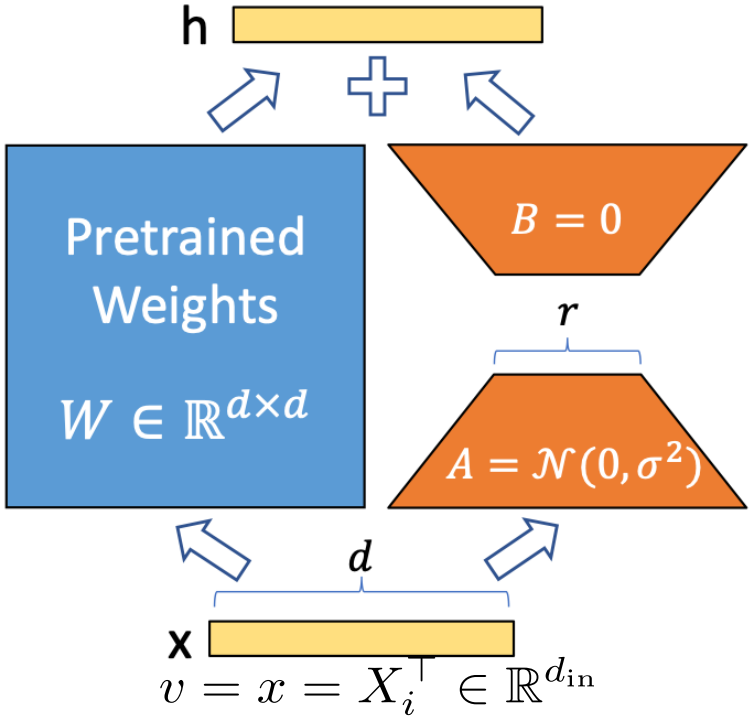


PEFT also applies to preference and reinforcement-learning objectives.

[21] Hu et al., Low-Rank Adaptation (LoRA), ICLR 2022.

Low-Rank Adaptation (LoRA)

LoRA freezes the base matrix and learns two small adapter matrices.



$$h = \left(W + \frac{\alpha}{r} BA\right) v$$

$$W \in \mathbb{R}^{d \times d}, \quad A \in \mathbb{R}^{r \times d}, \quad B \in \mathbb{R}^{d \times r}$$

r adapter rank α update scaling

The base matrix is frozen.

The adapter matrices are trained.

The SFT loss stays unchanged.

Low rank limits adaptation capacity while reducing trainable parameters.

[21] Hu et al., ICLR 2022.

LoRA: Shapes and Parameter Count

Use column vectors, as in the original LoRA figure. The frozen base is W.

$$W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \qquad A \in \mathbb{R}^{r \times d_{\text{in}}}, \qquad B \in \mathbb{R}^{d_{\text{out}} \times r}$$

$$\underbrace{r(d_{\text{in}} + d_{\text{out}})}_{\text{trainable adapter parameters}} \qquad \text{versus} \qquad \underbrace{d_{\text{in}} d_{\text{out}}}_{\text{full matrix parameters}}$$

$$\text{Full:} \quad 4096^2 = 16{,}777{,}216$$

$$\text{LoRA:} \quad 8(4096 + 4096) = 65{,}536 = \frac{16{,}777{,}216}{256}$$

Example: a 4,096 × 4,096 matrix with rank r = 8 uses 256× fewer trainable values.

[21] Hu et al., LoRA, ICLR 2022.

Why Initialize One LoRA Factor to Zero?

Random A and zero B preserve the pretrained function at initialization.

$$s = \frac{\alpha}{r}, \qquad W_{\text{eff}} = W + sBA, \qquad G = \frac{\partial \mathcal{L}}{\partial W_{\text{eff}}}$$

$$\nabla_A \mathcal{L} = sB^\top G, \qquad \nabla_B \mathcal{L} = sGA^\top$$

Random A, B = 0
ΔW = 0; learning can start.

A = 0, B = 0
Both adapter gradients are zero.

[21] Hu et al., ICLR 2022.

Why Quantize the Frozen Base?

LoRA saves gradient and optimizer memory, but the frozen base still occupies GPU memory.
Those weights are still needed to compute activations during fine-tuning.

Method	Frozen base format	Raw storage: 7 billion weights
LoRA	16-bit BF16	14 GB
QLoRA	4-bit NF4	3.5 GB

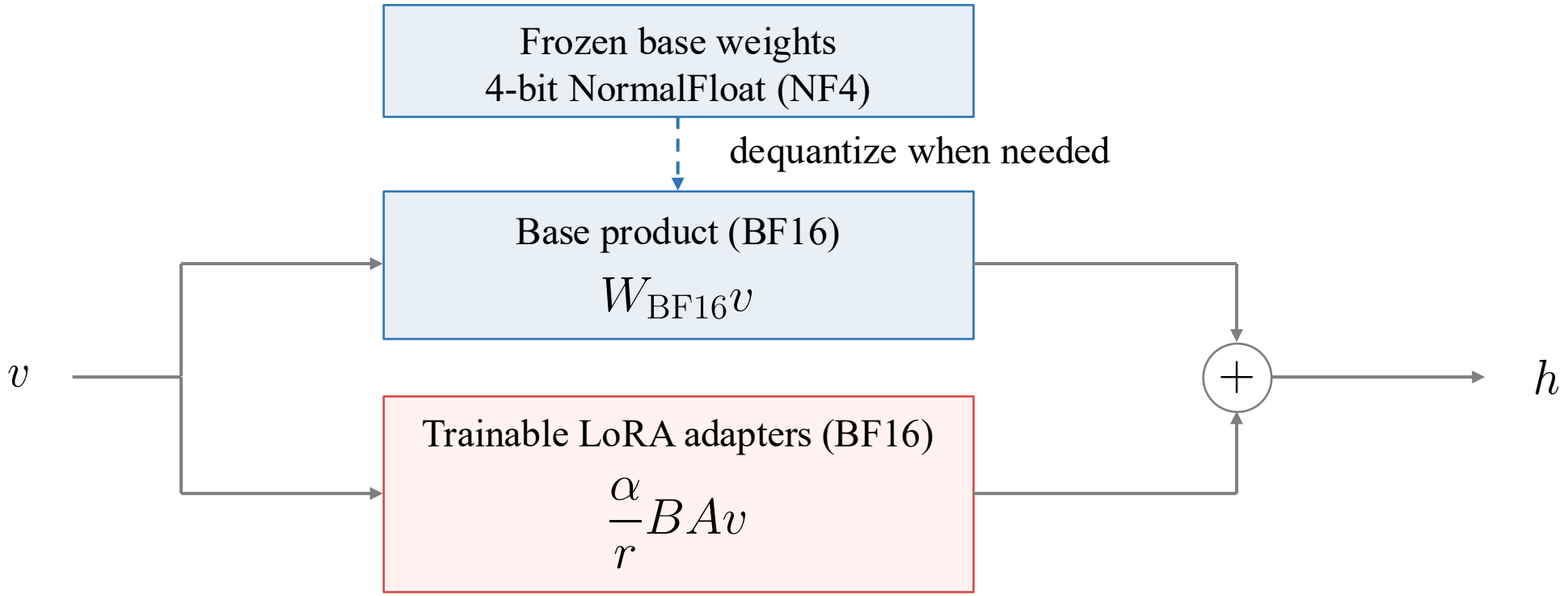
Quantization stores approximate weights with fewer bits. Keep the small adapters trainable.

QLoRA reduces frozen-base storage as well as trainable state [22].
This makes fine-tuning larger pretrained models more accessible with limited GPU memory.

[22] Dettmers et al., NeurIPS 2023. Decimal GB, raw base weights only. Metadata, adapters, activations and buffers add memory.

Quantized LoRA (QLoRA)

Store the frozen base in 4 bits. Dequantize to BF16 when computing.



Training: backpropagate through both paths. Update only the adapters.

Inference: both paths use fixed weights. No backward pass or optimizer state.

[22] Dettmers et al., NeurIPS 2023, §3. Dashed blue arrow: weight reconstruction. Gray arrows: activations.

References

[1] Hugging Face LLM Course. Tokenizers. [source]

[2] Sennrich et al. (2016). Neural Machine Translation of Rare Words with Subword Units. ACL. [source]

[3] OpenAI. tiktoken educational implementation: byte-level BPE training and encoding. [source]

[4] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, 2003. [source]

[5] OpenAI. tiktoken: encoding definitions and model mappings. [encodings] [model mappings]

[6] OpenAI Cookbook. How to count tokens with tiktoken. [source]

[7] Google Research. BERT: WordPiece tokenizer implementation. [source]

[8] Kudo (2018). Subword Regularization: Improving Neural Network Translation Models. ACL. [source]

[9] Kudo and Richardson (2018). SentencePiece. EMNLP; official toolkit and T5 benchmark. [paper] [toolkit]

[10] Touvron et al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. [source]

[11] Llama Team (2024). The Llama 3 Herd of Models. arXiv:2407.21783. [source]

References

[12] Xue et al. (2022). ByT5: Towards a Token-Free Future with Pre-Trained Byte-to-Byte Models. TACL. [source]

[13] MiMo Team (2026). MiMo-V2-Flash Technical Report. arXiv:2601.02780v2. [source]

[14] Brown et al. (2020). Language Models are Few-Shot Learners. NeurIPS. [source]

[15] Kaplan et al. (2020). Scaling Laws for Neural Language Models. [source]

[16] Hoffmann, J., et al., 2022. Training compute-optimal large language models. NeurIPS, 35. [source]

[17] Wei et al. (2022). Emergent Abilities of Large Language Models. TMLR. [source]

[18] Schaeffer et al. (2023). Are Emergent Abilities of Large Language Models a Mirage? NeurIPS. [source]

[19] Fedus et al. (2022). Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. JMLR. [source]

[20] Ouyang et al. (2022). Training Language Models to Follow Instructions with Human Feedback. NeurIPS. [source]

[21] Hu et al. (2022). LoRA: Low-Rank Adaptation of Large Language Models. ICLR. [source]

[22] Dettmers et al. (2023). QLoRA: Efficient Finetuning of Quantized LLMs. NeurIPS. [source]