CMPT 413/713: Natural Language Processing

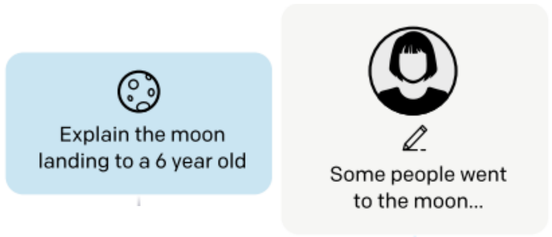
LLM data and scaling

Spring 2026

2026-03-09

Some slides adapted from Anoop Sarkar

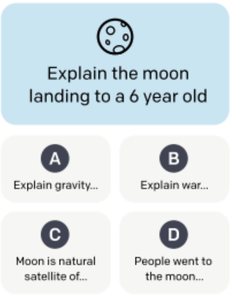
Training stages



Piles of unlabeled text!

Text with "instructions" and "responses"

Human preference data
Data about what people prefer



Self-supervised training LM objective

$p(w_t \mid w_{<t})$

Pre-training
LM training on large, large amount of data

Pre-training can be broken into stages (mid-training)

Supervised training LM objective

$p(w_t \mid w_{<t}; \text{prompt})$

Instruction-tuning
Supervised fine-tuning for instructions

Reinforcement learning

Preference optimization
Align to human preferences

Post-training
(Can have more iterations)

Post-training for reasoning

- RLHF - align to human preferences
- RLVR - align to verifiable rewards (for reasoning)

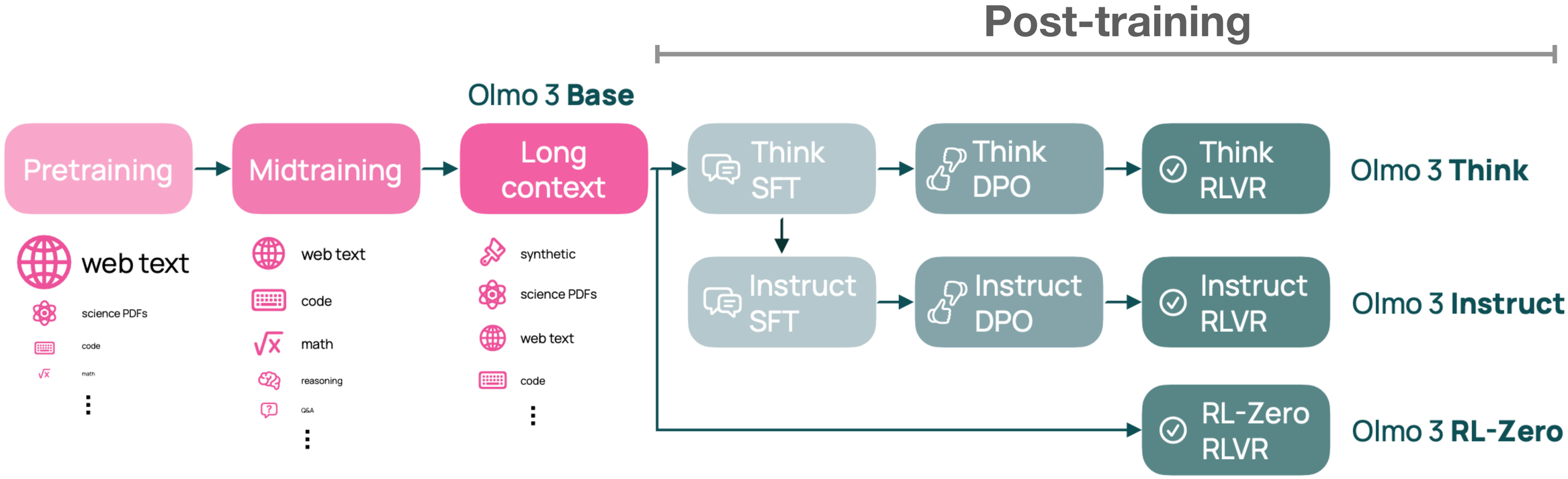


Figure 2 Depiction of model flow for Olmo 3. Development is divided into major **base model training (left)** and **post-training (right)** stages, each further divided into sub-stages with their own recipes (i.e., training data and method).

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Reinforcement learning from verifiable rewards

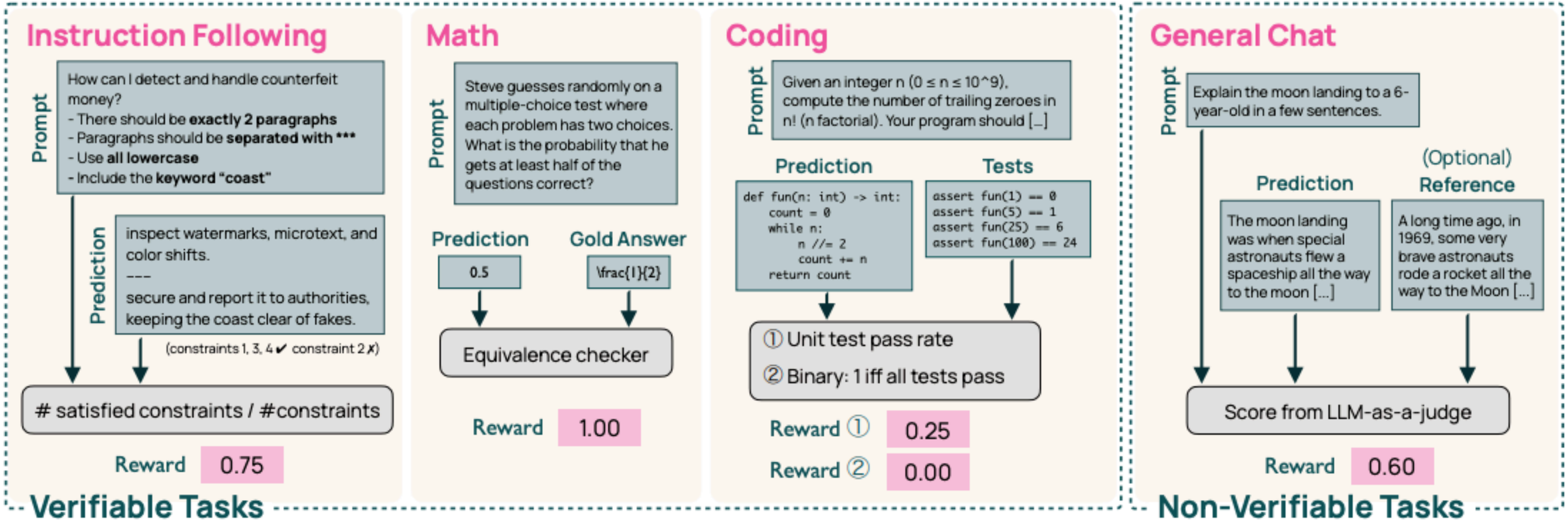


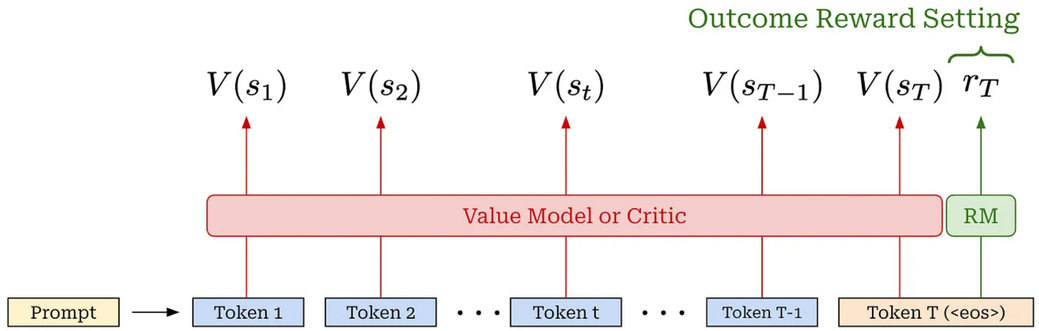
Figure 16 Verifiers and reward design for verifiable and non-verifiable tasks.

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Post-training

Alignment

- RLHF - aligns to human feedback
- Usually uses PPO
  - Requires a critic model to compute the advantage



Reasoning

- RLVR - aligns to verifiable reward
- Usually uses GRPO
  - No need for critic model
  - Get relative advantage for a group

Group Relative Advantage for GRPO

$$\hat{A}_{i,t} = \frac{r_i - \mathtt{mean}(\mathbf{r})}{\mathtt{std}(\mathbf{r})}$$

t-th token in i-th output

Prompt (or question) sampled from dataset: $x \sim \mathcal{D}$

Group of sampled outputs (from old policy): $\mathbf{y} = \{y_1, y_2, \ldots, y_G\}$

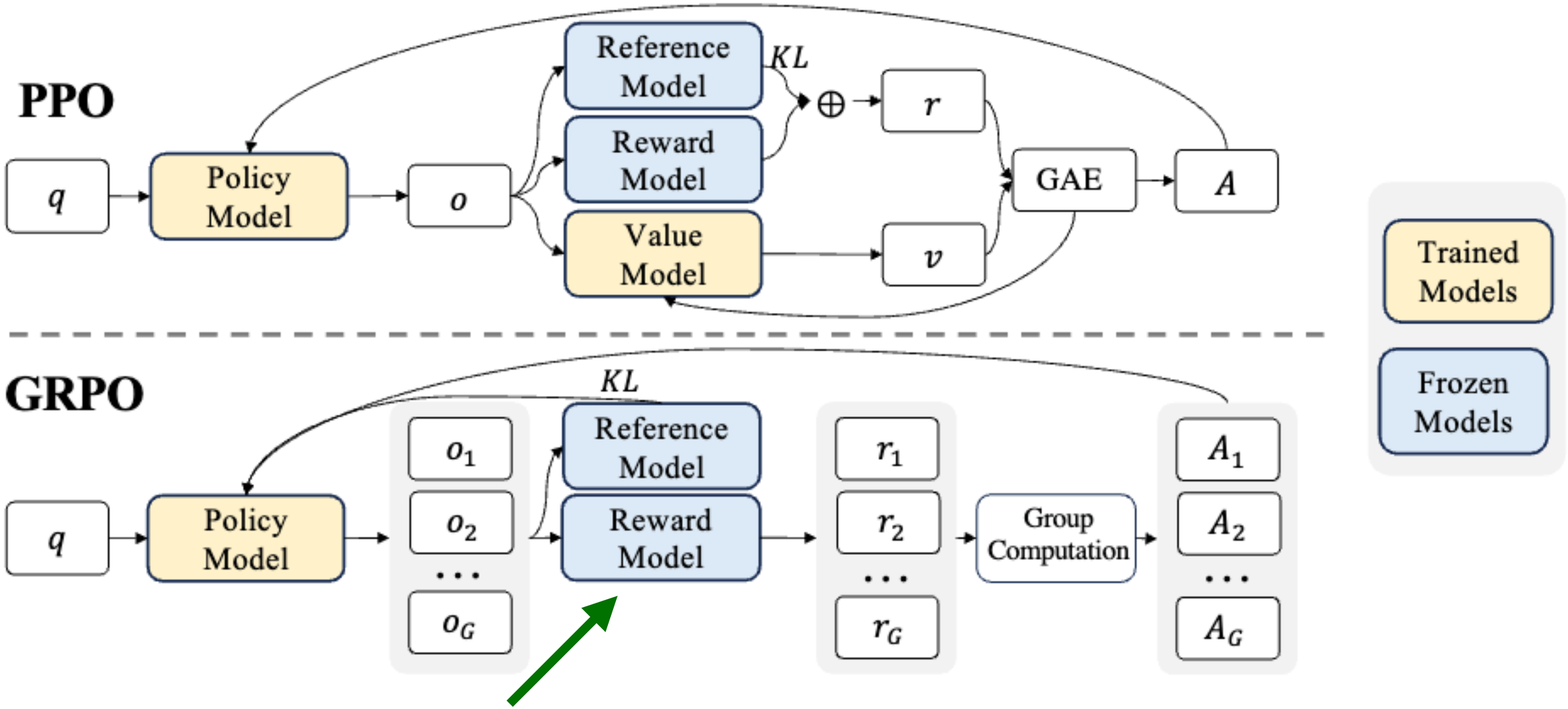
Outcome reward for each output: $\mathbf{r} = \{r_1, r_2, \ldots, r_G\}$

https://cameronrwolfe.substack.com/p/grpo

- Need lots of samples (completions) per prompt to get good estimate of advantage

Advantage - difference between estimated expected reward for action (action-value / Q function) and estimated reward over all actions (value function)

GRPO vs PPO



Neural reward model can be replace by rule-based reward model in RLVR

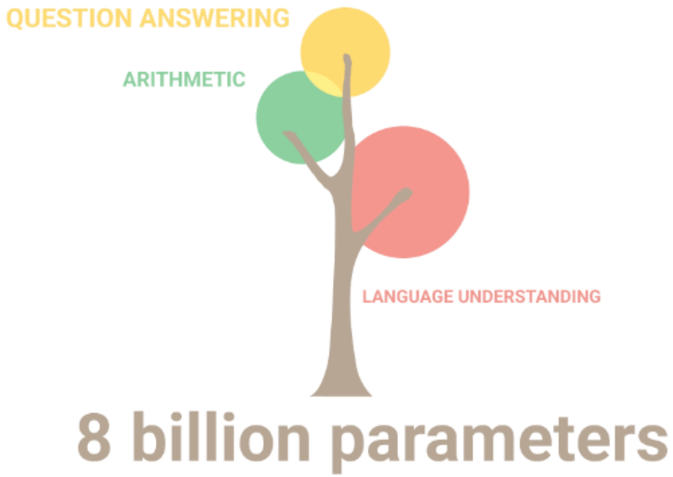
https://arxiv.org/pdf/2402.03300

Eliminate value model

Large models

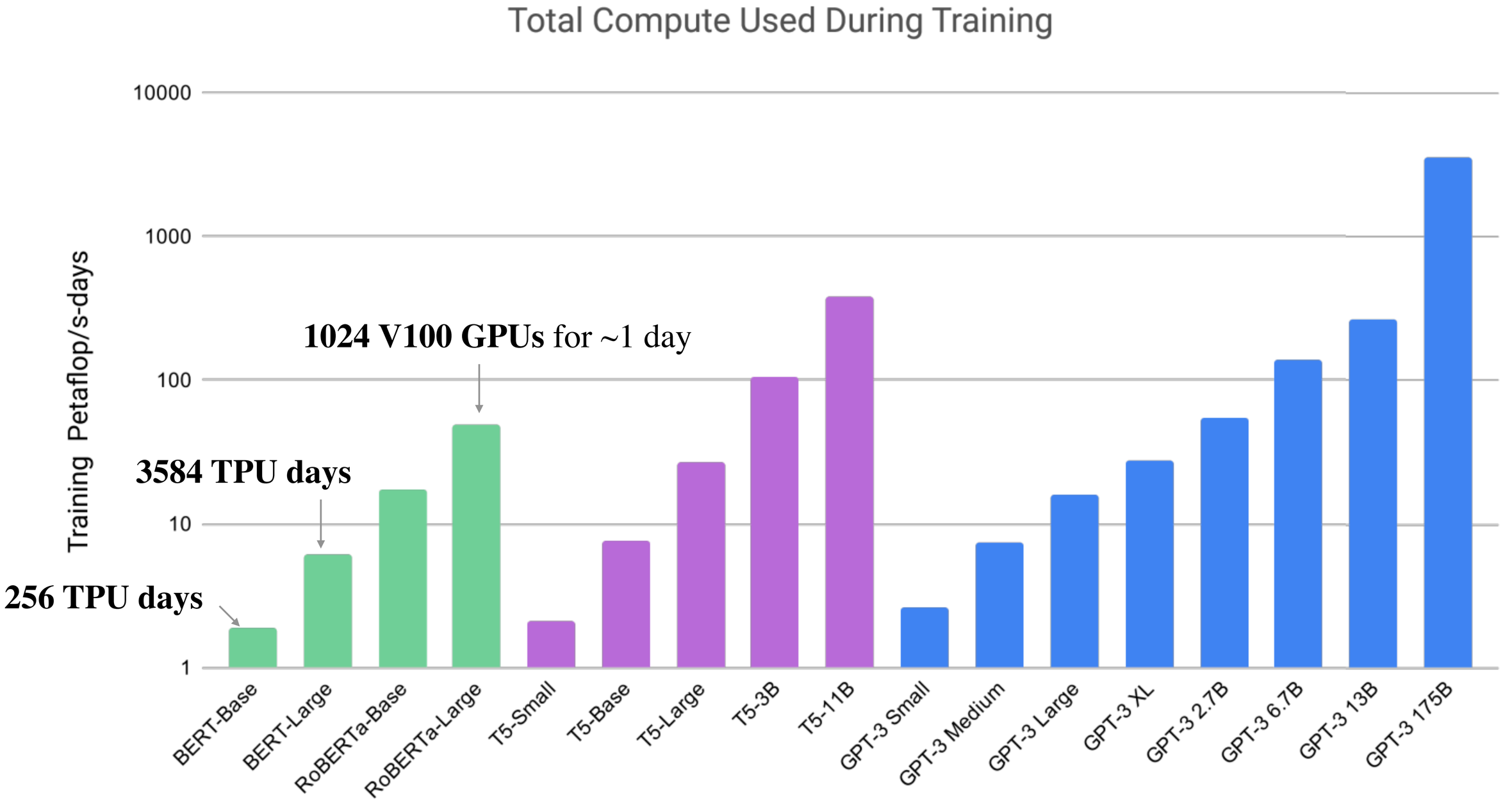
New capabilities emerge at scale

How to train these large models?



How to train small models with similar capabilities?

https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html



Language Models are Few-Shot Learners (Brown et al. 2020)

Olmo training phases

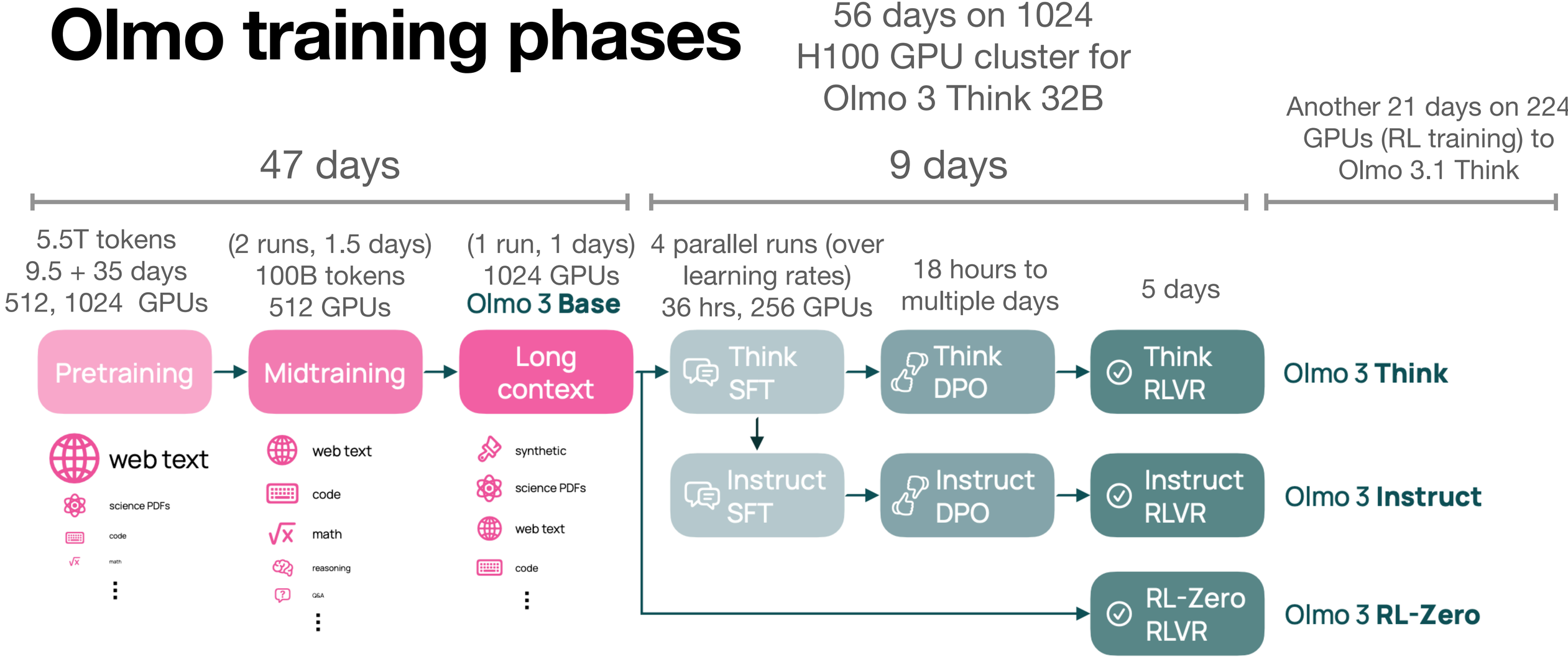


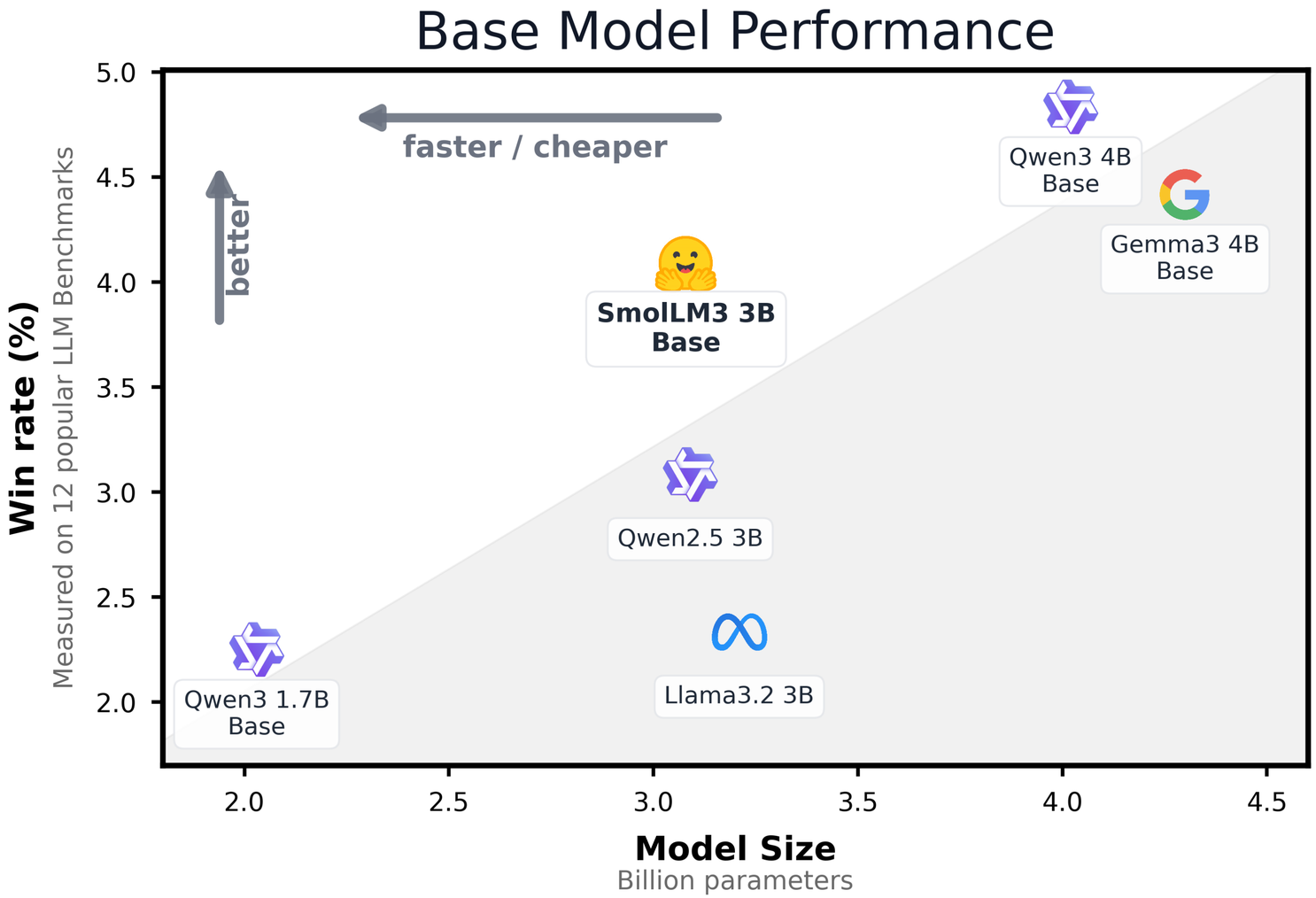
Figure 2 Depiction of model flow for Olmo 3. Development is divided into major **base model training (left)** and **post-training (right)** stages, each further divided into sub-stages with their own recipes (i.e., training data and method).

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- Training process
- Infrastructure

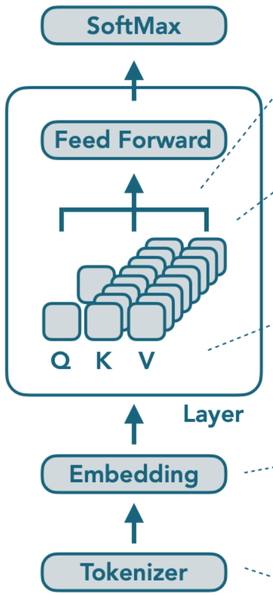


https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- **Model architecture**
- Data
- Training process
- Infrastructure



Grouped Query Attention

The 16 attention heads share 4 query which maintains the performance of full multi-head attention and saves memory during inference.

Intra-Document Masking

Attention masking ensures that tokens from different documents in the same training sequence don't attend to each other which helps with long context training.

NoPE

We use NoPE which selectively removing rotary position embeddings from every 4th layer improving long context performance.

No Weight Decay in Embeddings

We remove weight decay from embedding layers to improve training stability as embedding norms naturally stabilize at healthier values.

Multilingual Tokenizer

We use the Llama 3.2 tokenizer which covers all languages used for pretraining.

Training Configuration

Parameter count: 3.08B
Initialization: N(0, std=0.02)
Layers: 36
Rope theta: 50k

Sequence length: 4096
Batch size: 2.36M Tokens
Optimizer: AdamW (eps=1e-8, beta1=0.8, beta2=0.95)
Learning rate (peak): 2e-4
Gradient Clipping: 1.0
Weight decay: 0.1
Gradient accumulation: 1
Micro batch size: 3
Precision: bf16
Tensor parallel: 2
Data parallel: 192

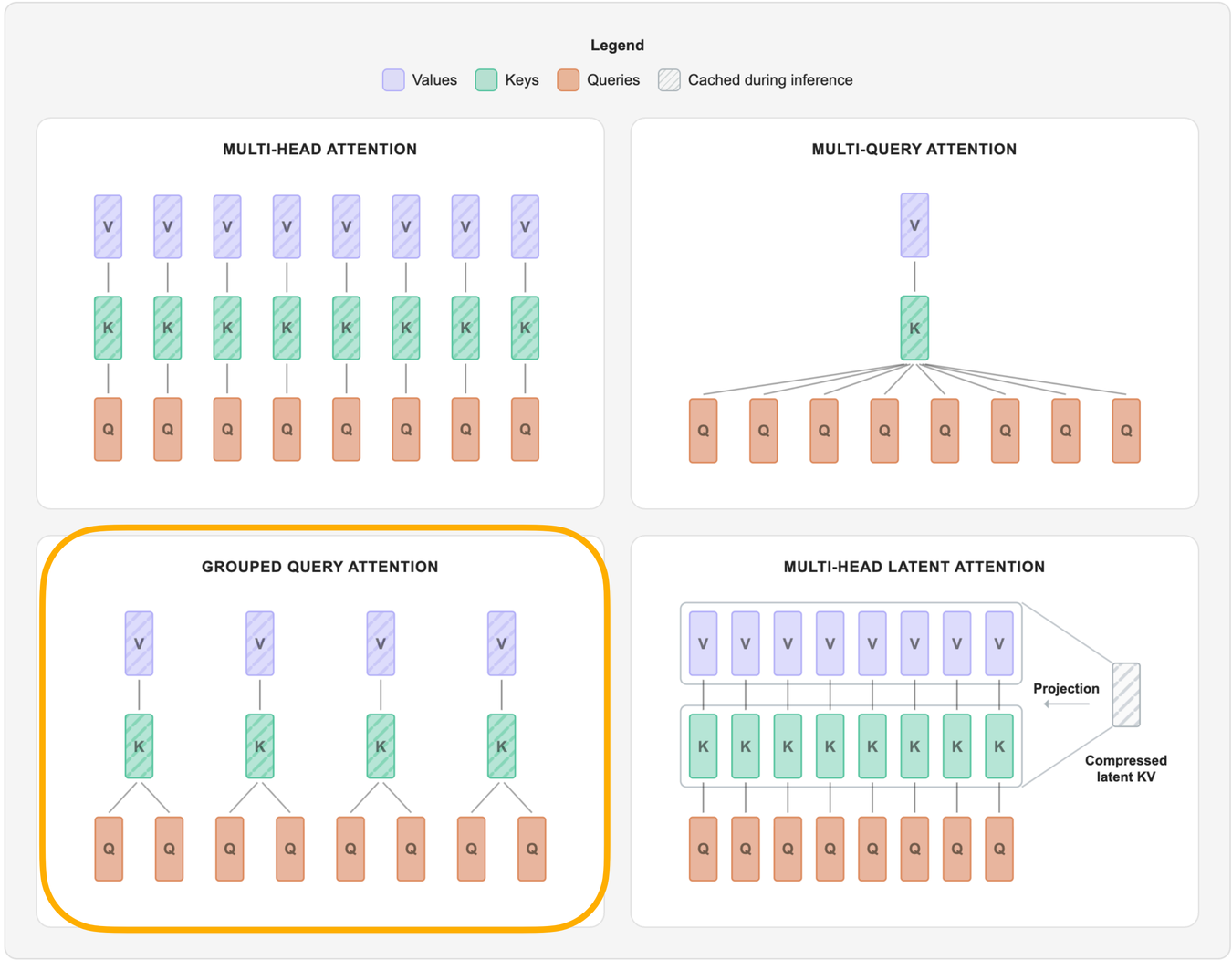
Throughput: 14k tokens/sec/gpu
MFU: 29.43 %
Training duration: 24 days

https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- **Ablations**
- Model architecture
- Data
- Training process
- Infrastructure

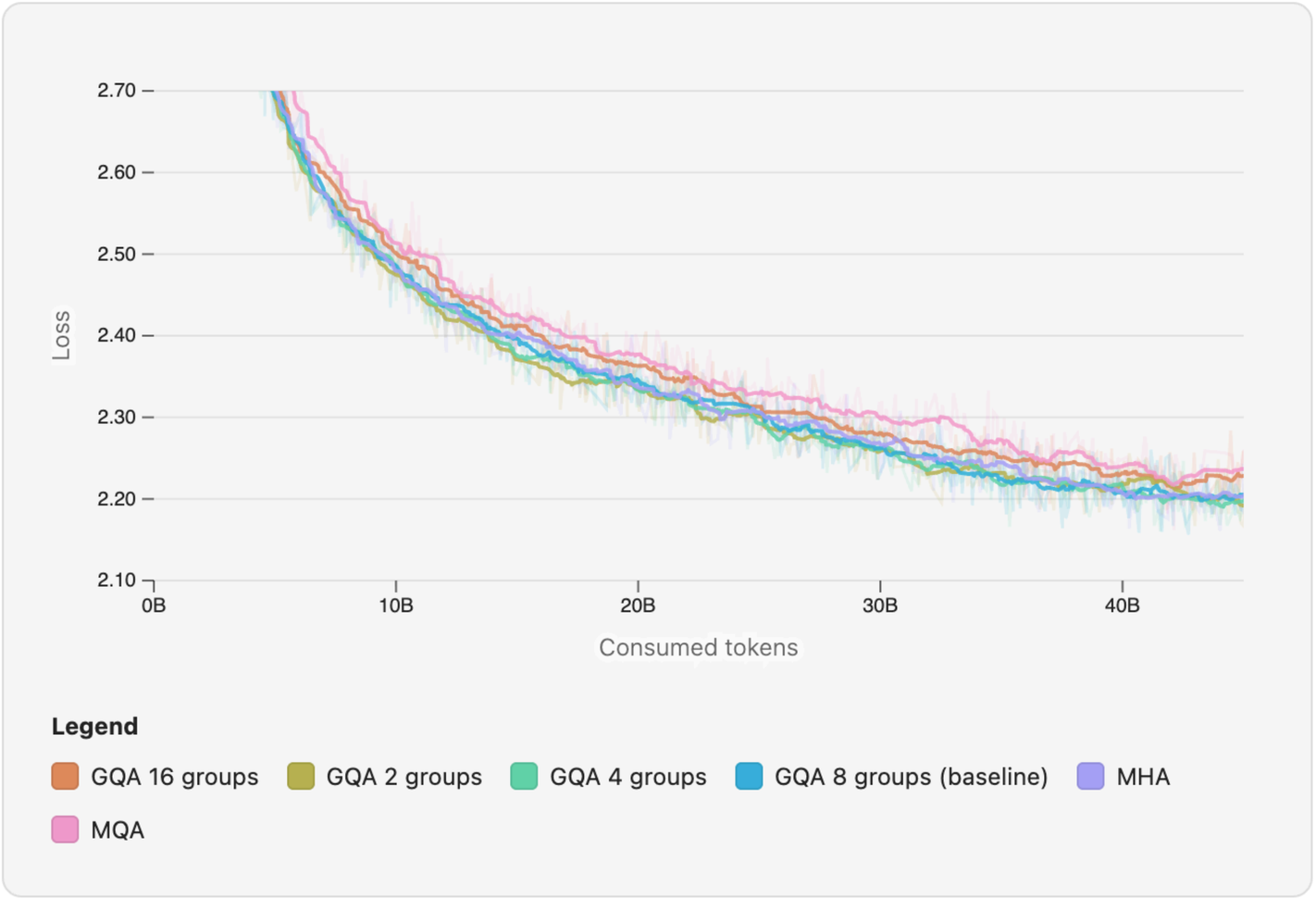


HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- **Ablations**
- Model architecture
- Data
- Training process
- Infrastructure

Grouped Query Attention with 2, 4, or 8 groups performed similar to MHA
SmolLM3 used 4 groups

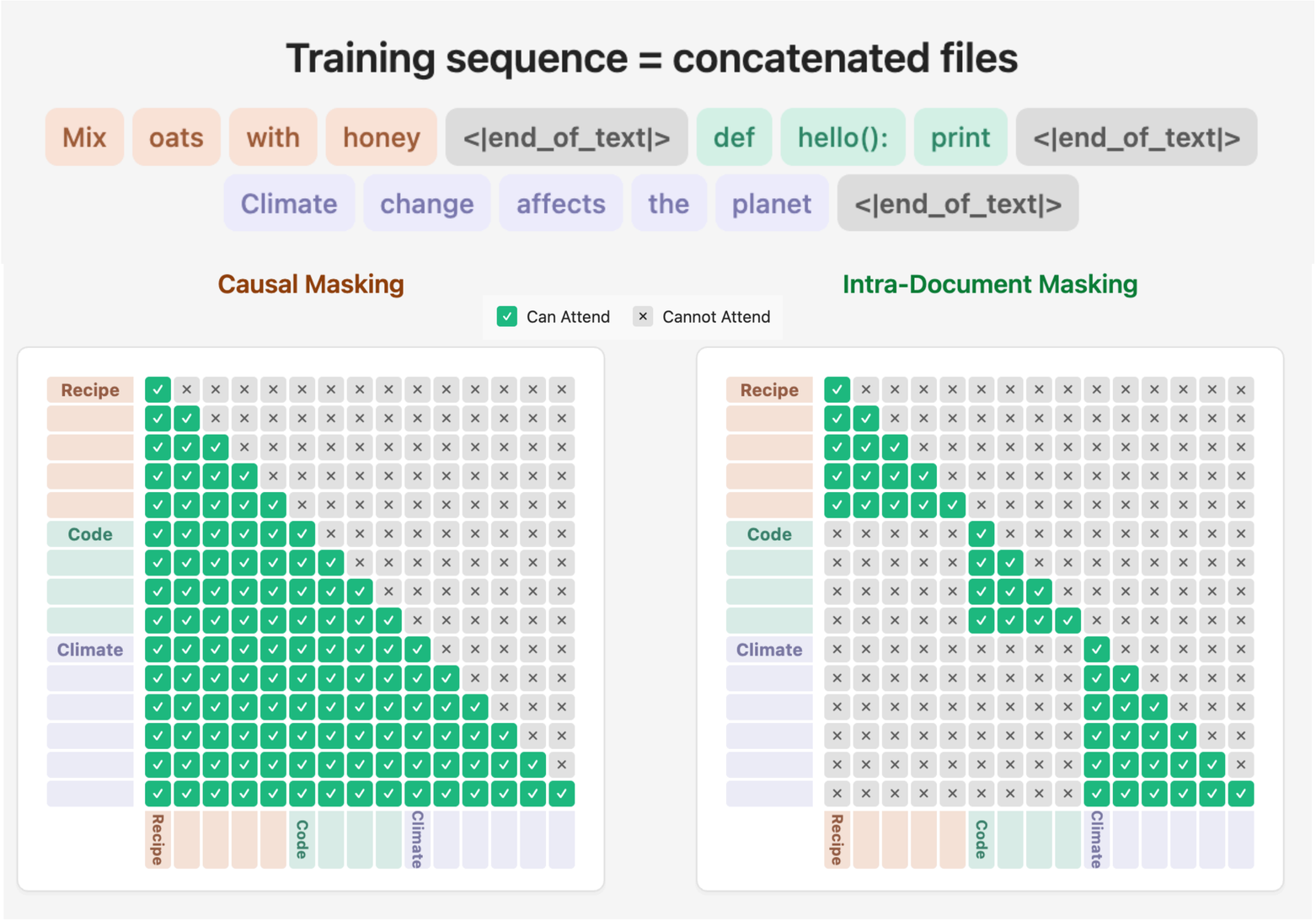


HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- Training process
- Infrastructure

Document packing and masking

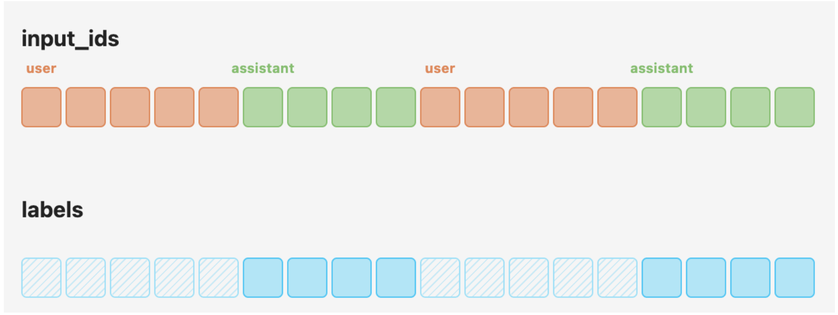


HuggingFace guide to pretraining

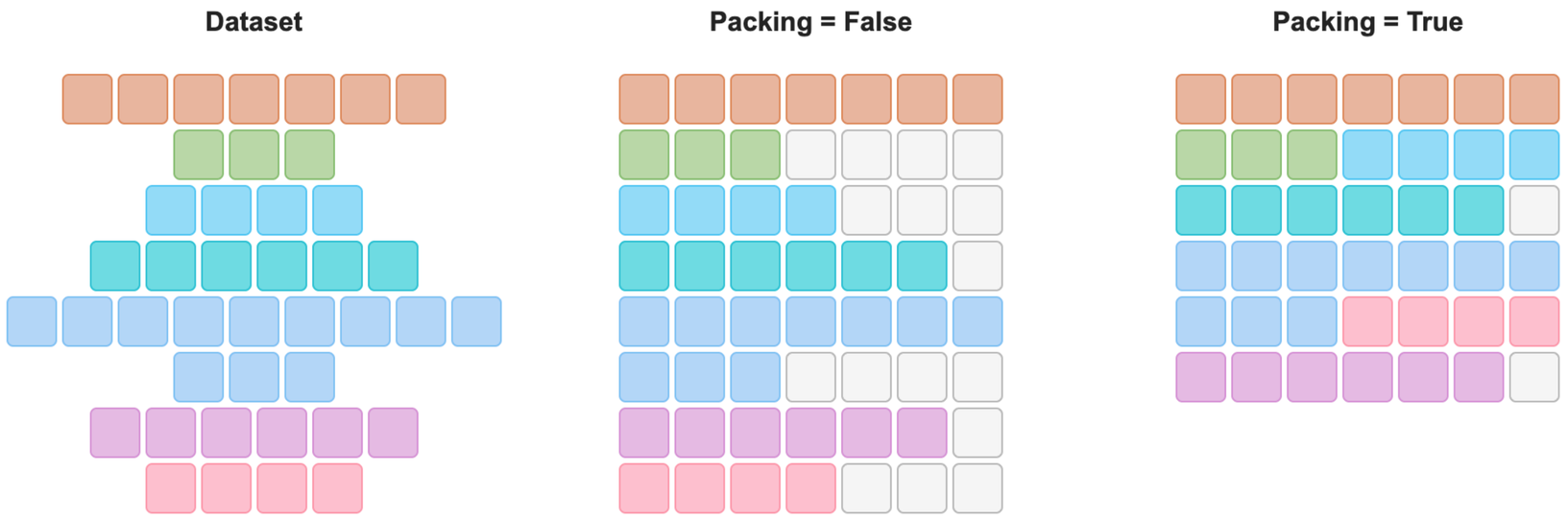
https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- Training process
- Infrastructure

Masking user prompts



Sequence packing

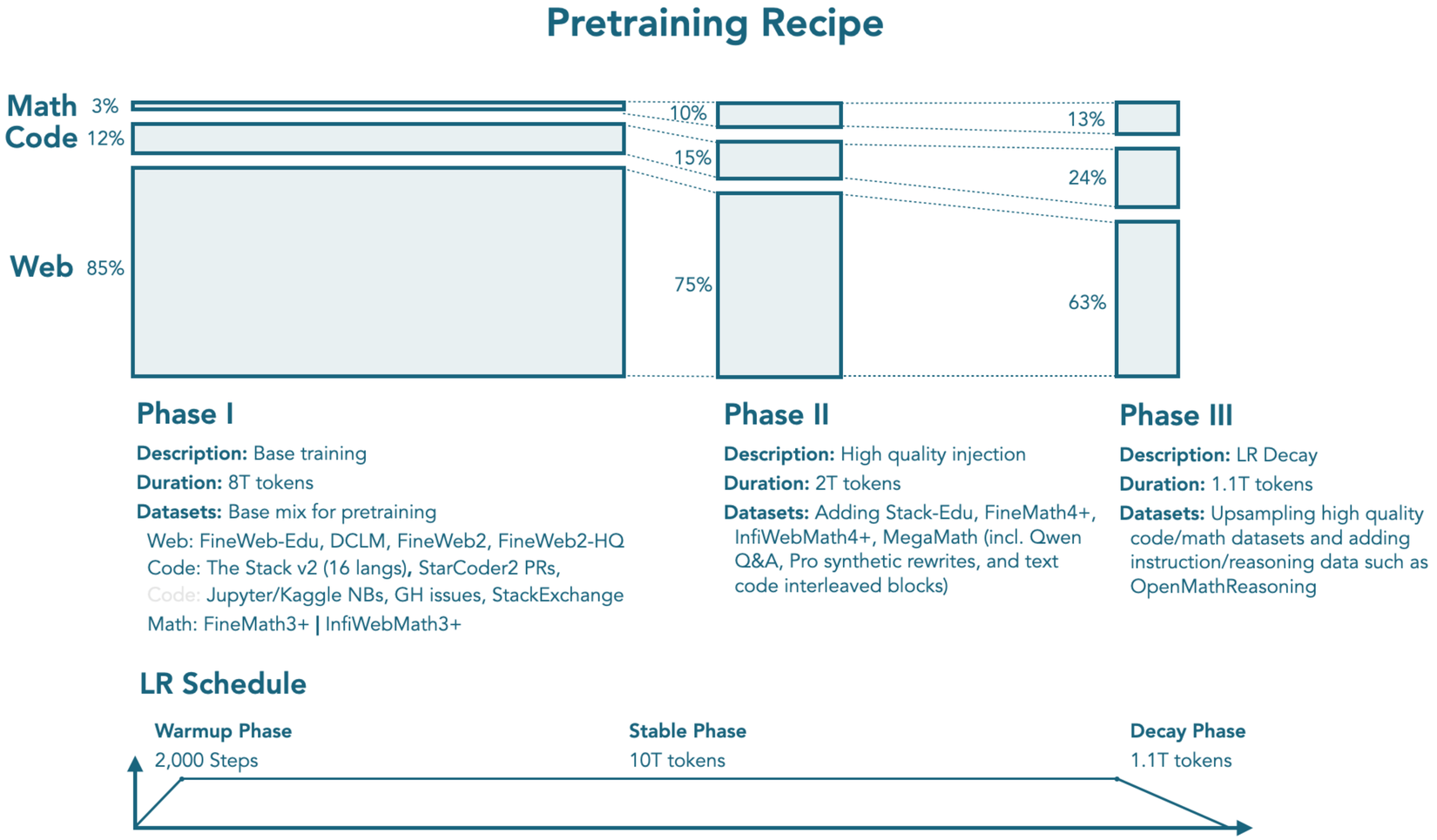


https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- **Training process**
- Infrastructure

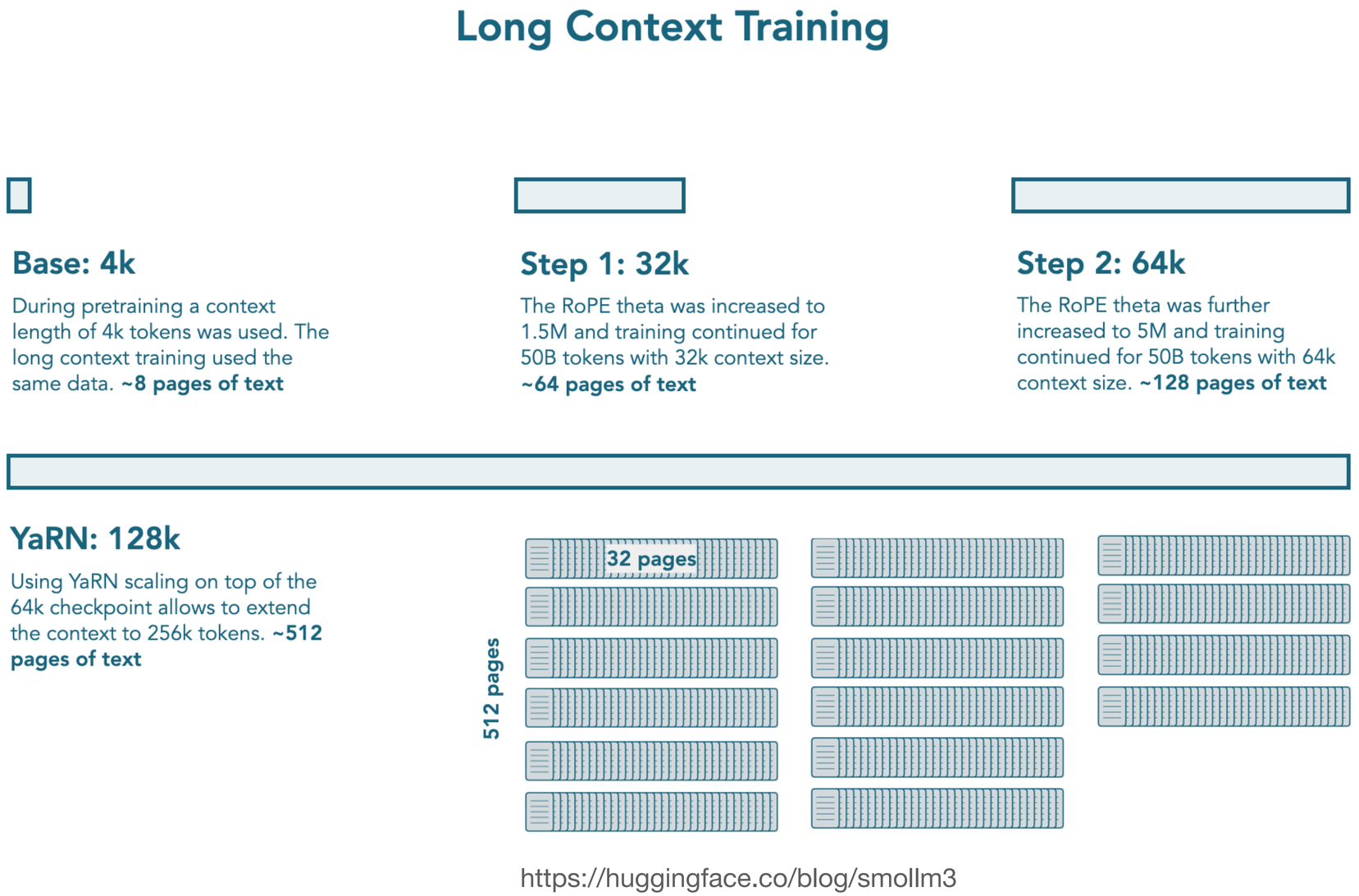


https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- **Training process**
- Infrastructure



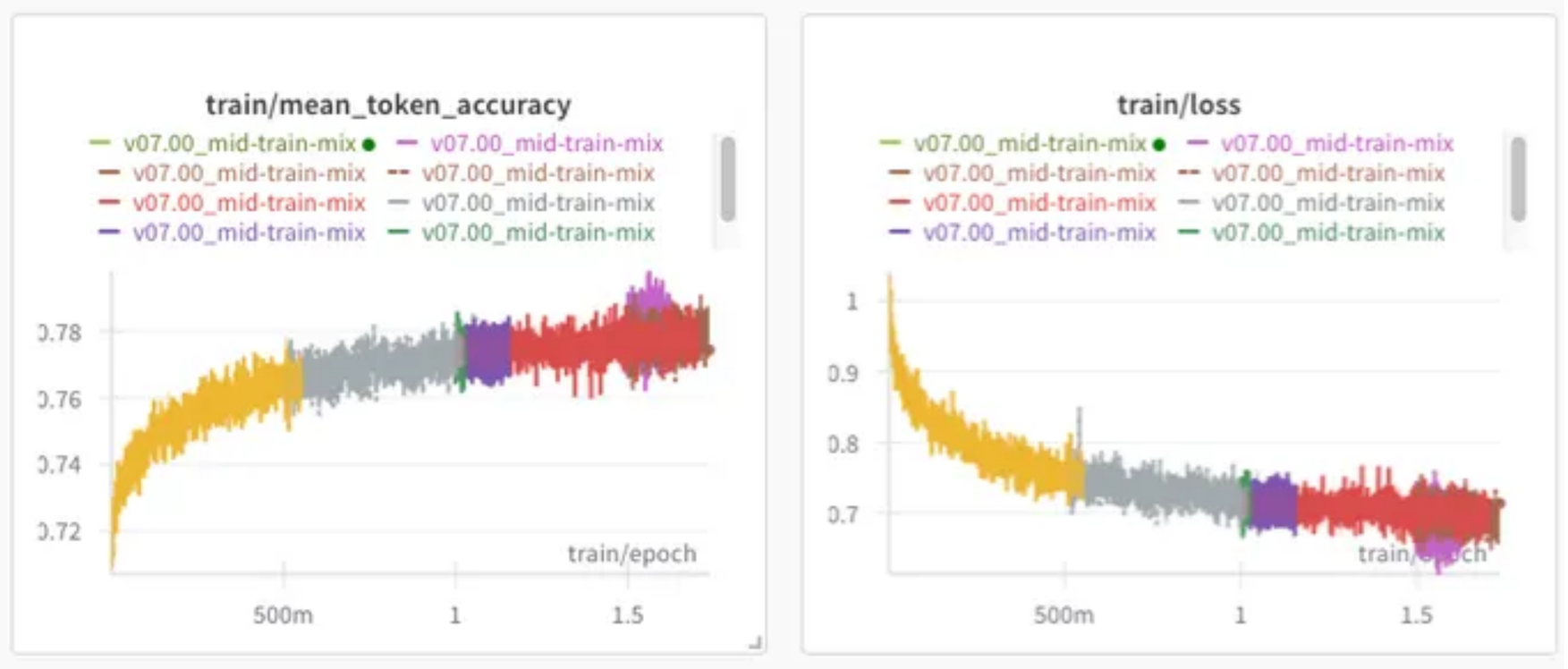
https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- **Training process**
- Infrastructure

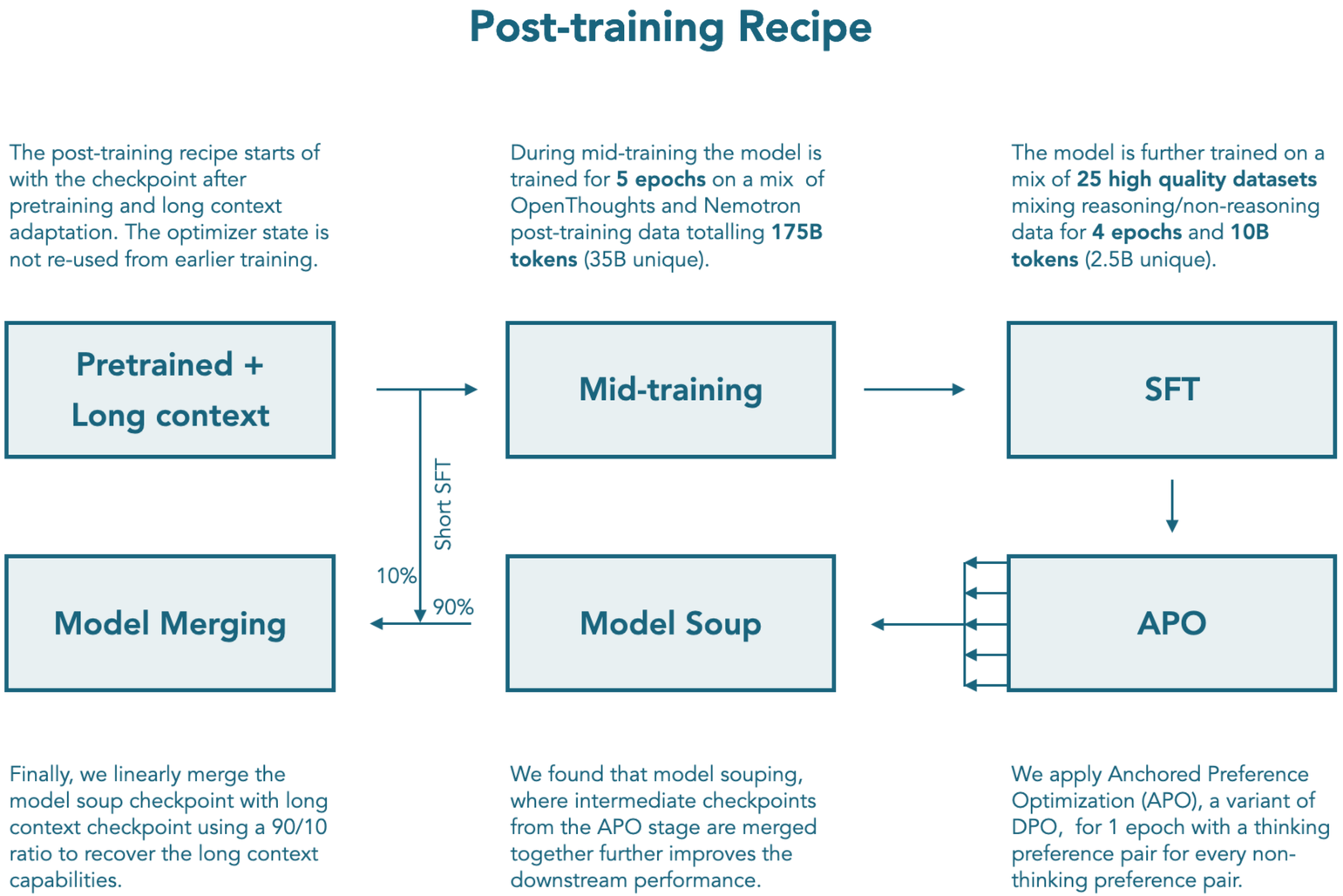
Train over weeks, restarts, monitor loss curve for spikes, fix instabilities, load data to local machine for training, make sure everything is properly checkpointed, …



HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- **Training process**
- Infrastructure



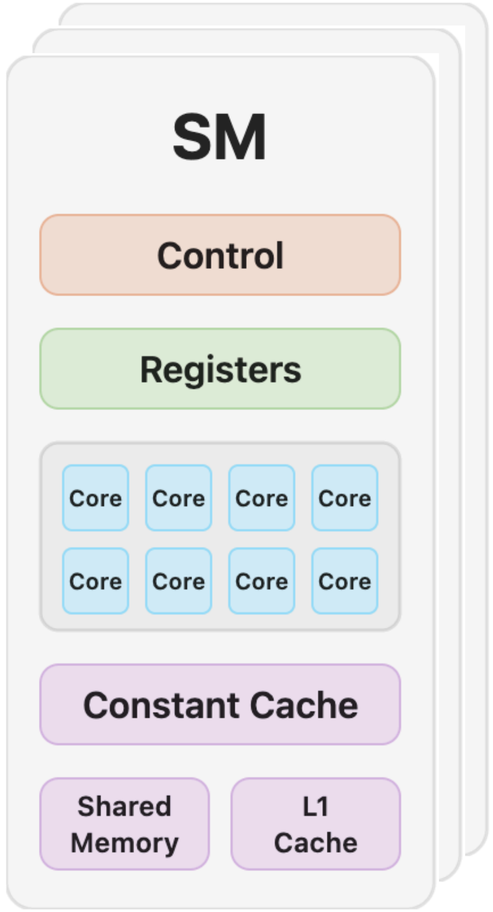
https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

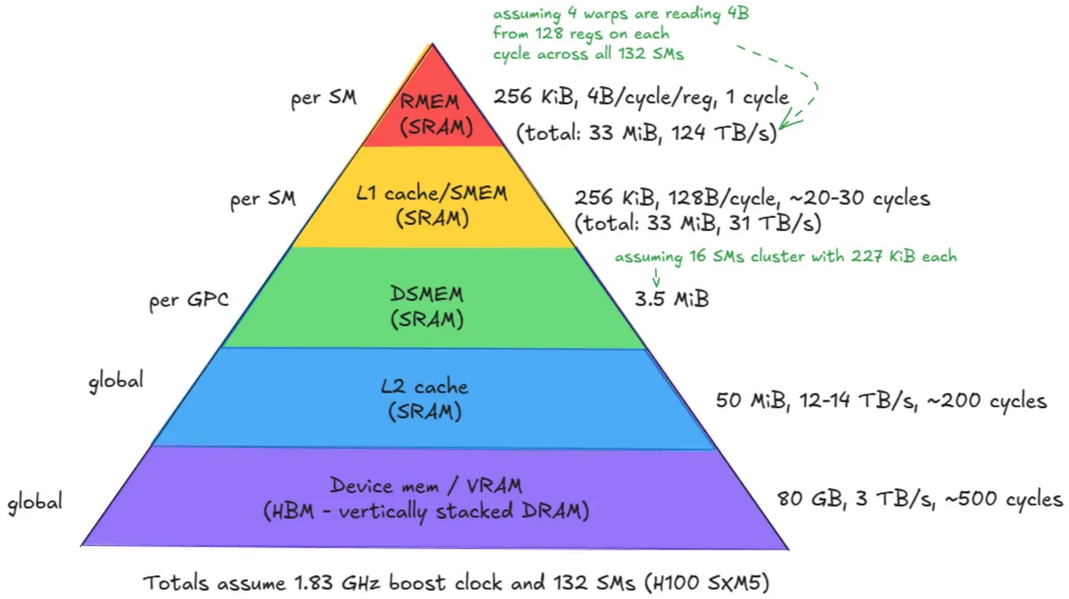
https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- Training process
- **Infrastructure**

GPUs have hundreds of Streaming Multiprocessors (SMs)
H100 contains 132 SMs



Need to get data into memory
Different memory speed and sizes



Memory hierarchy

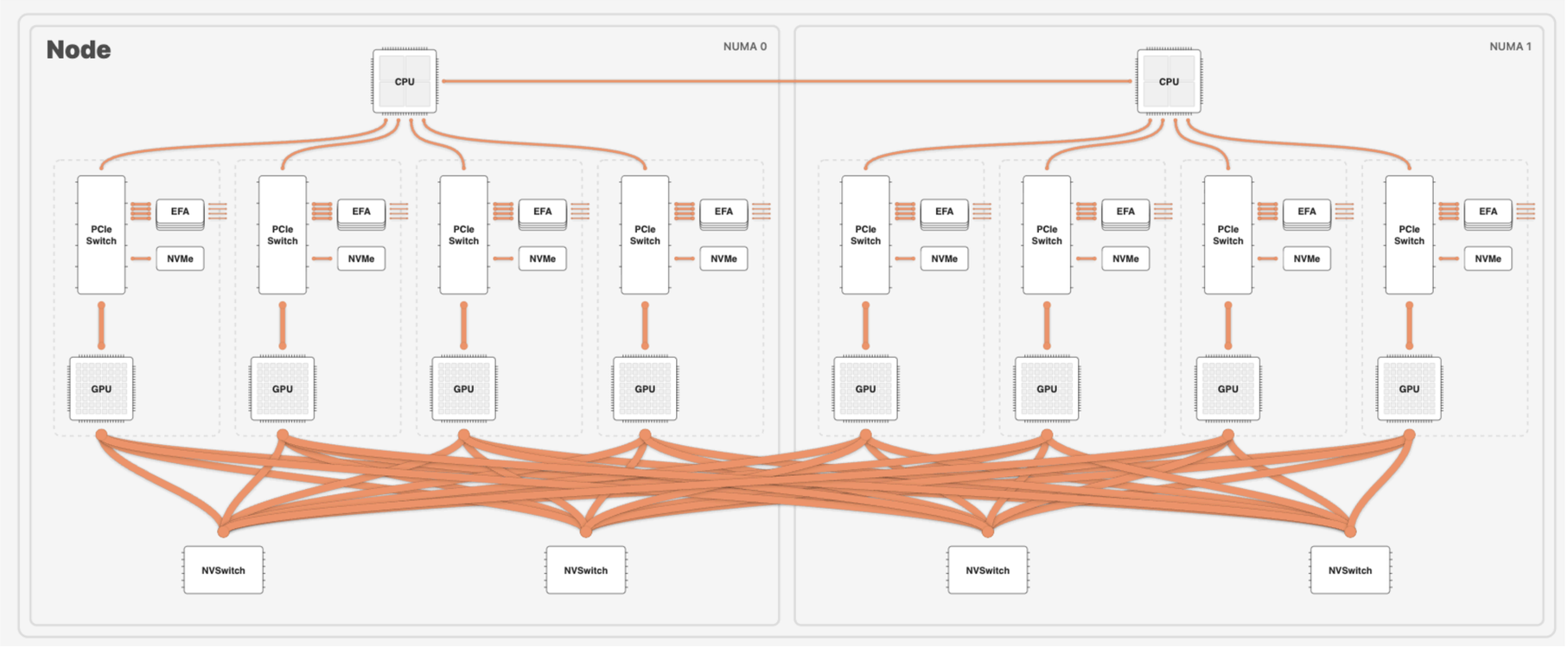
https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- Data
- Training process
- **Infrastructure**

Communicating beyond the GPU



https://huggingface.co/blog/smollm3

PaLM: Pathways data parallelism

- Trained on two TPU v4 pods
  - Each pod had 3072 TPU chips attached to 768 hosts (total 6144 chips)
  - Each pod had full copy of model parameters
- Model + data parallelism, no pipeline parallelism
  - 12-way model parallelism, 256-way data sharing

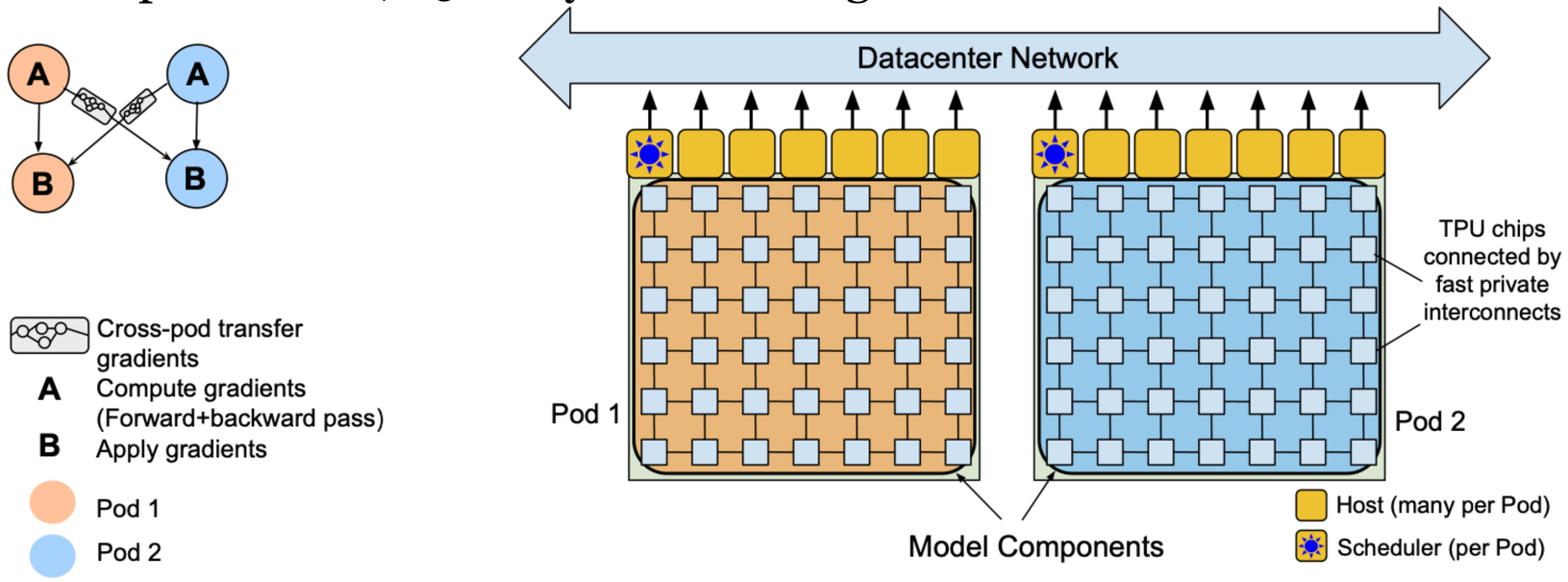
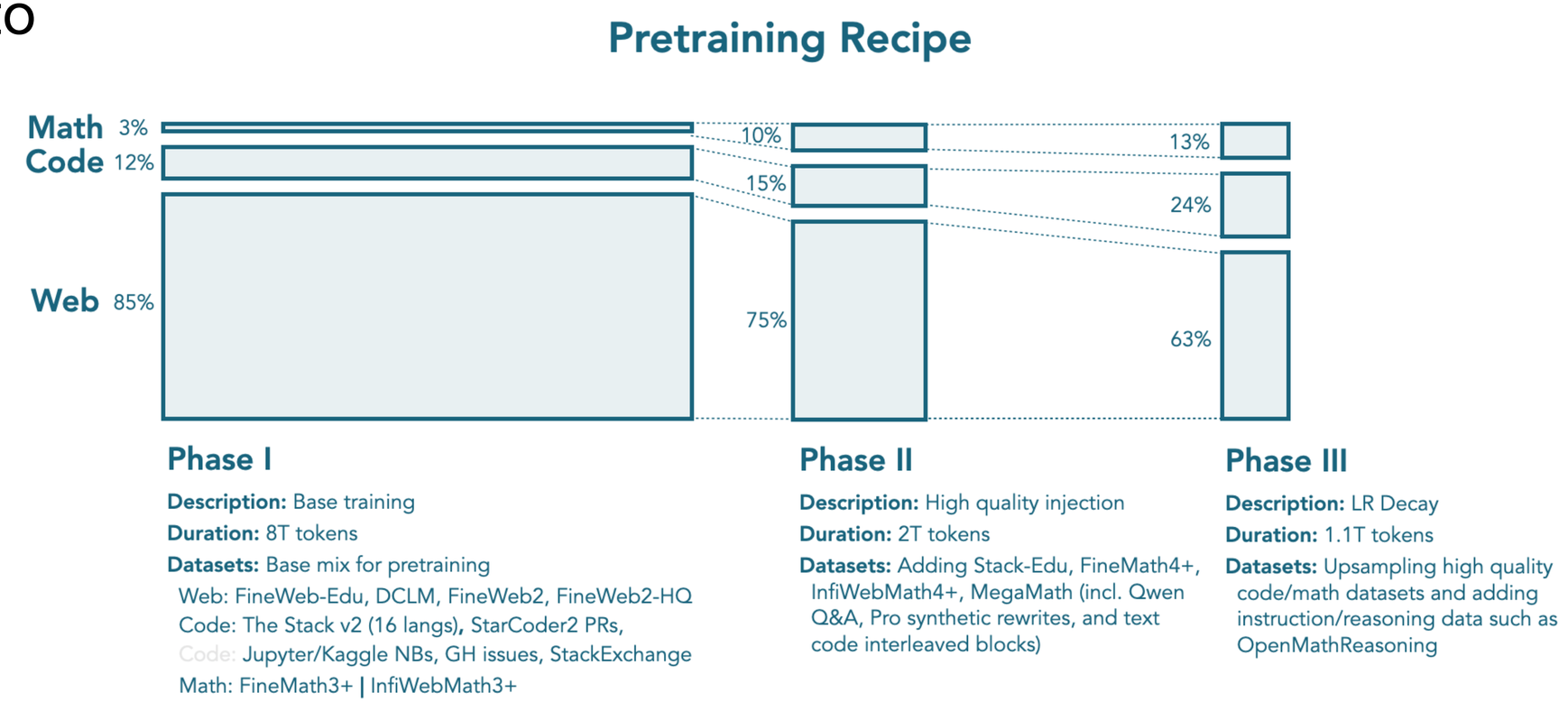


Figure 2: The Pathways system (Barham et al., 2022) scales training across two TPU v4 pods using two-way data parallelism at the pod level.

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- **Data**
- Training process
- Infrastructure



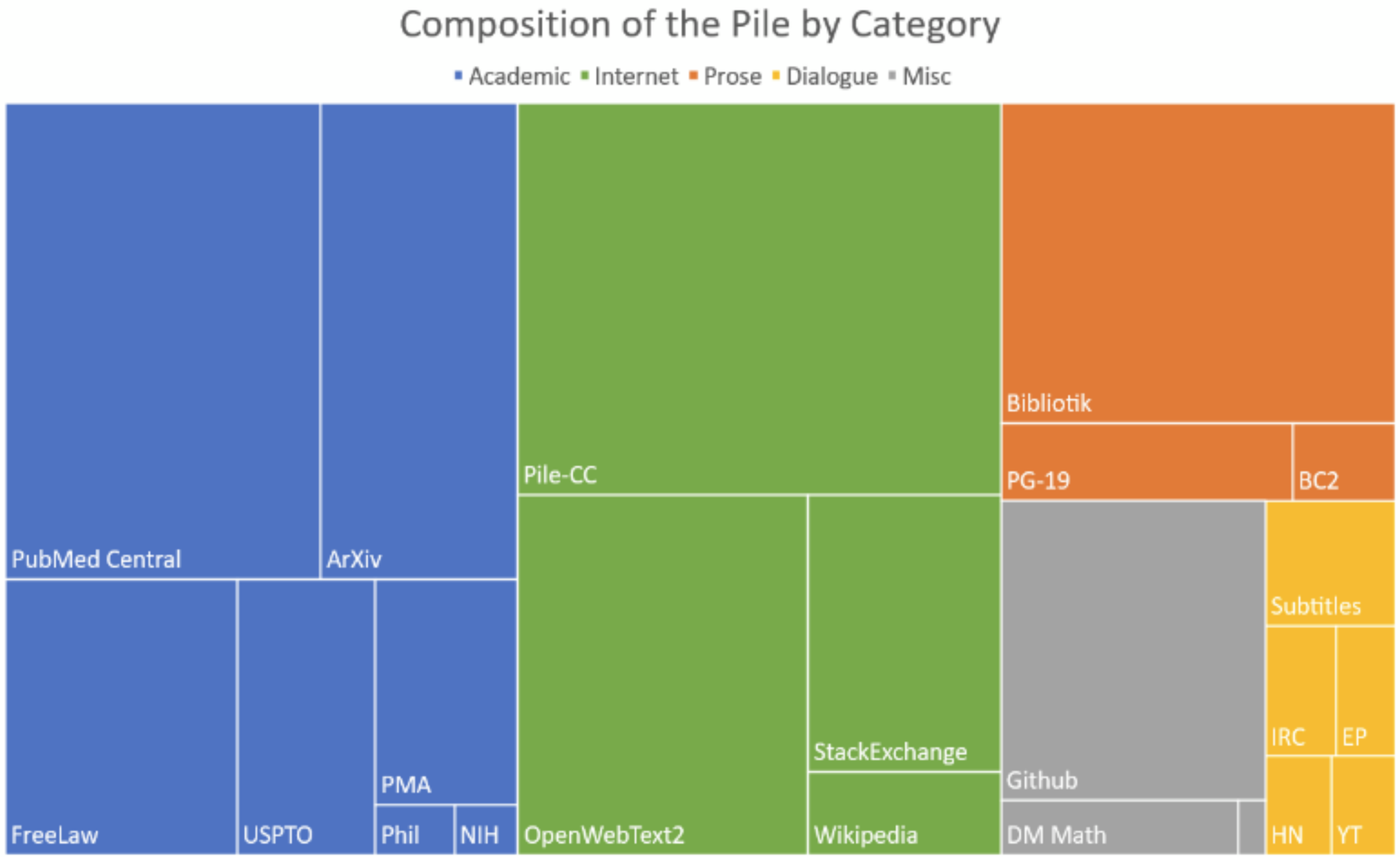
https://huggingface.co/blog/smollm3

Data for model training

The Pile

Open source data for language modelling

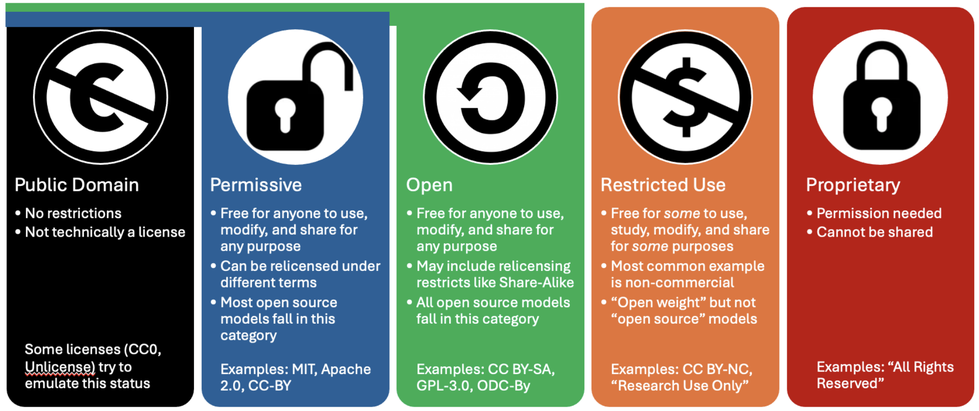
- 800 GB



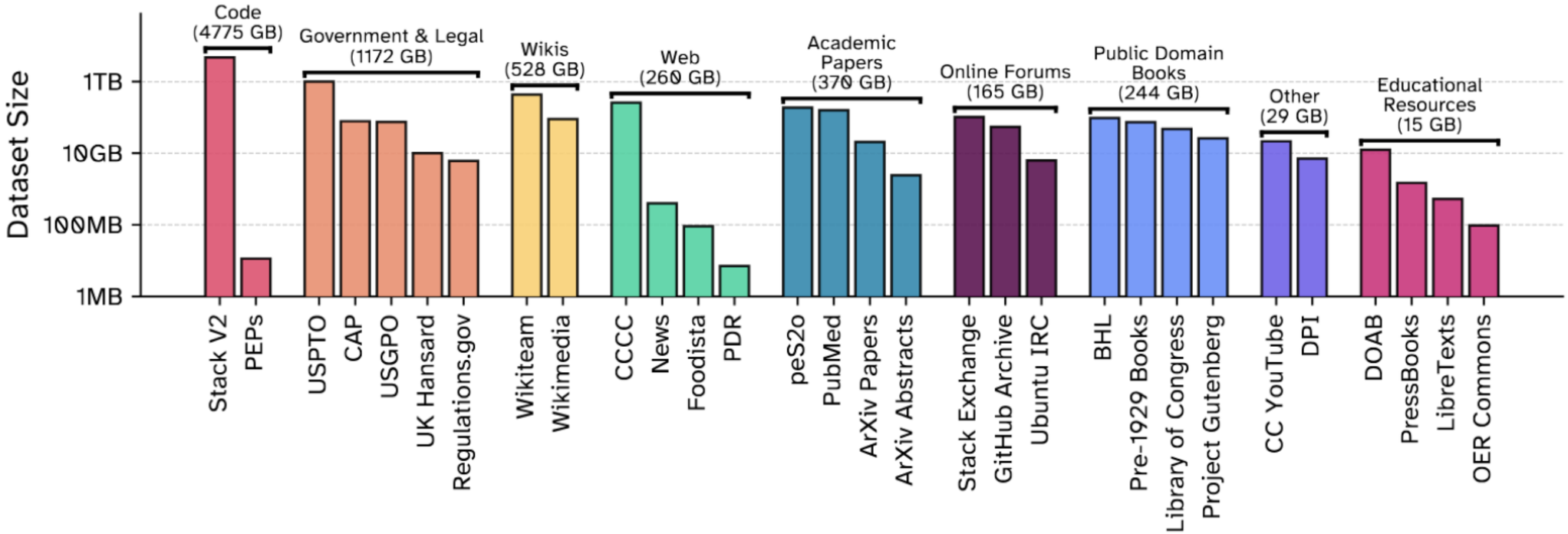
The Pile: An 800GB Dataset of Diverse Text for Language Modeling [Gao et al, EleutherAI, 2020]

The Common Pile (Comma)

Open source data for language modelling



- 8TB dataset - public and open license



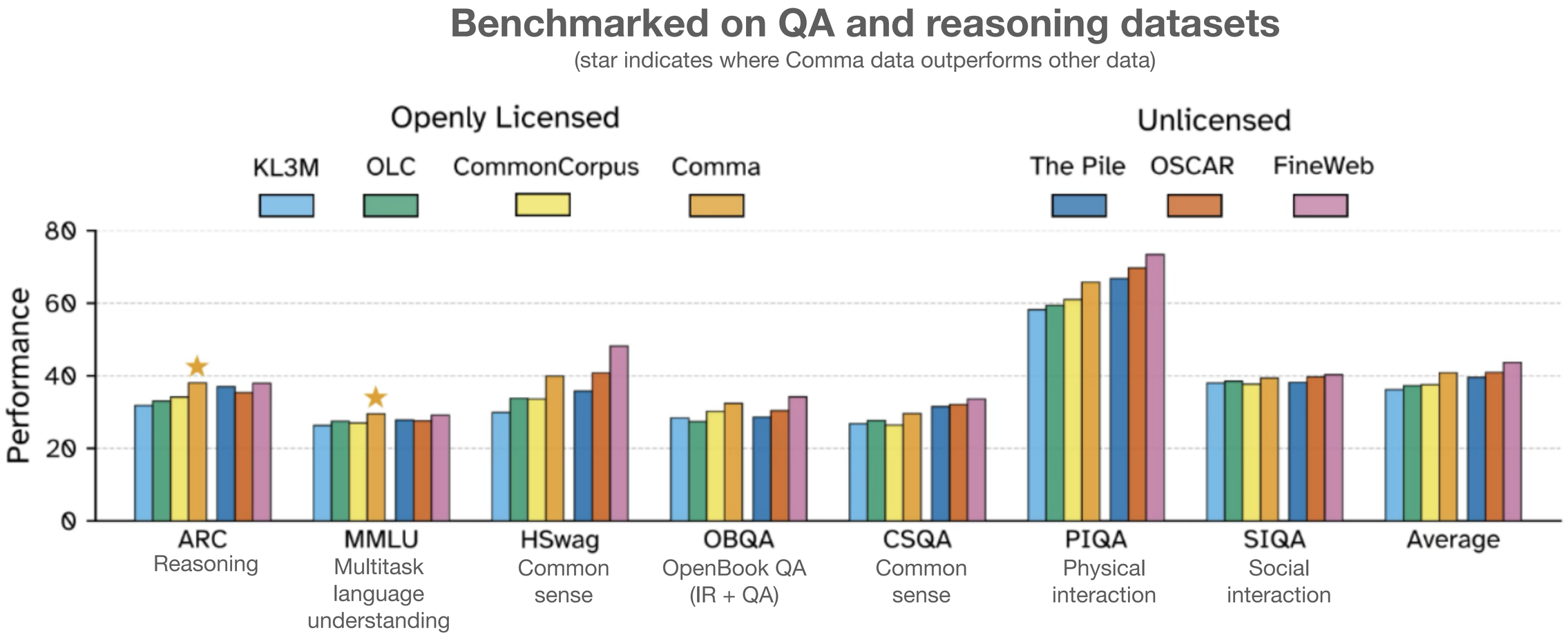
Collaboration across many institutions (Eleuther AI, HuggingFace, AI2, etc)

The Common Pile v0.1: An 8TB Dataset of Public Domain and Openly Licensed Text [Kandpal et al. 2025]

Comma v0.1

Open source data for language modelling

- Compare 1.7B model trained on different datasets (28B tokens)



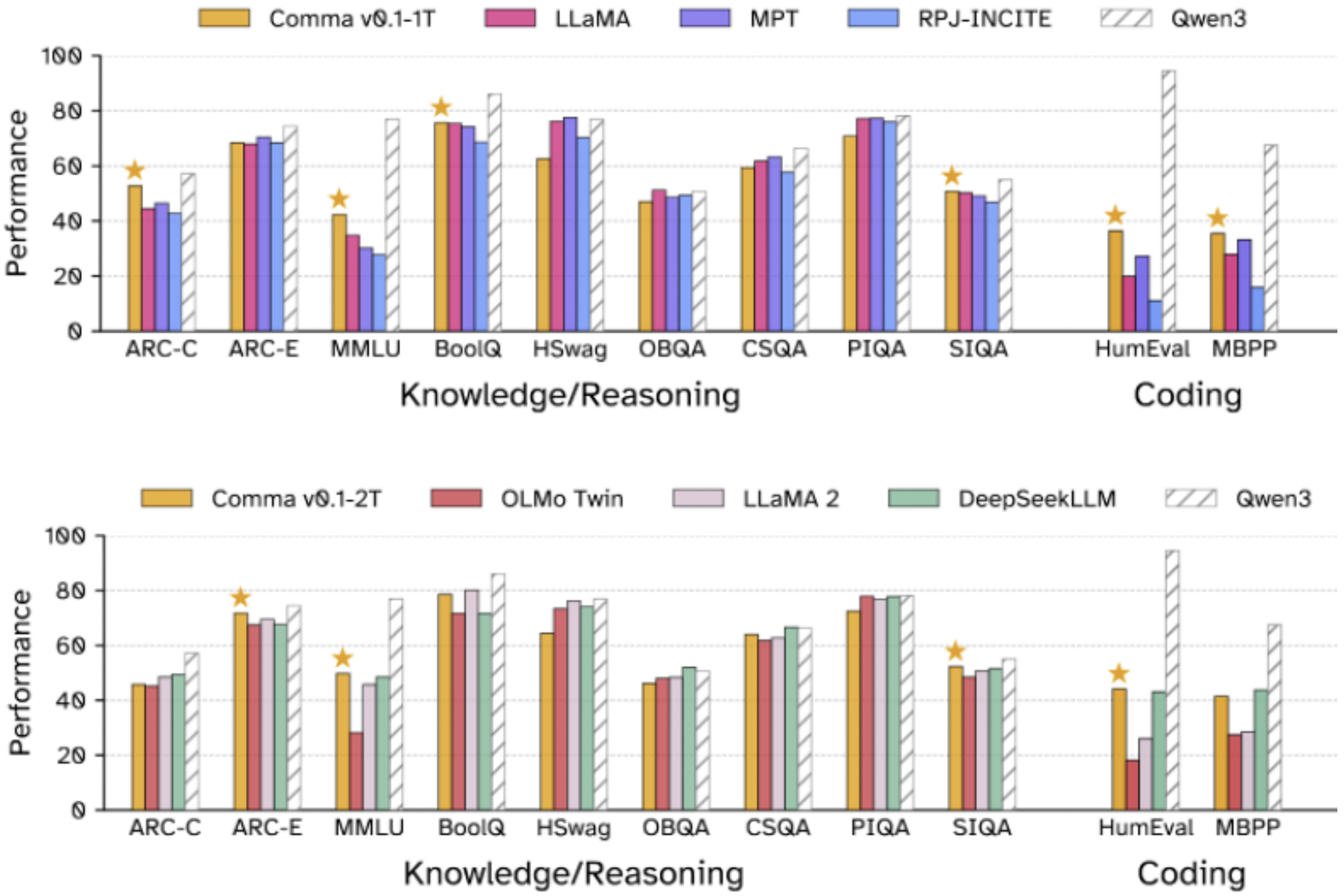
The Common Pile v0.1: An 8TB Dataset of Public Domain and Openly Licensed Text [Kandpal et al. 2025]

Comma v0.1

- 7B model trained on Comma v0.1
- BPE tokenizer with vocabulary size of 64K (trained on 600GB sample)
- Follow Llama-7B architecture and training

Comparison against similar size models

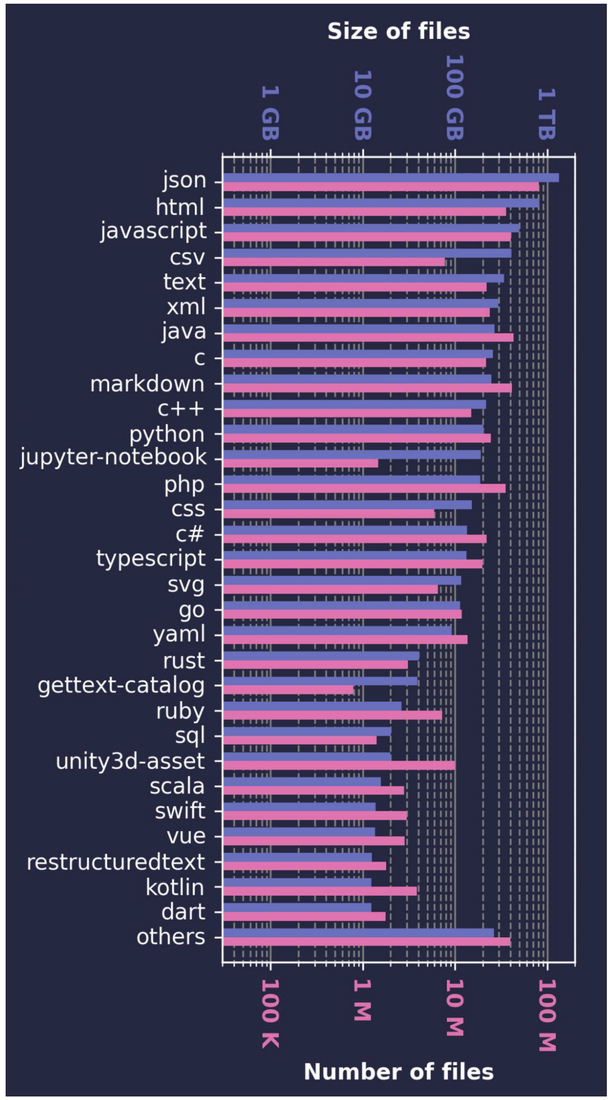
(star indicates where Comma model outperforms other compute-matched models)

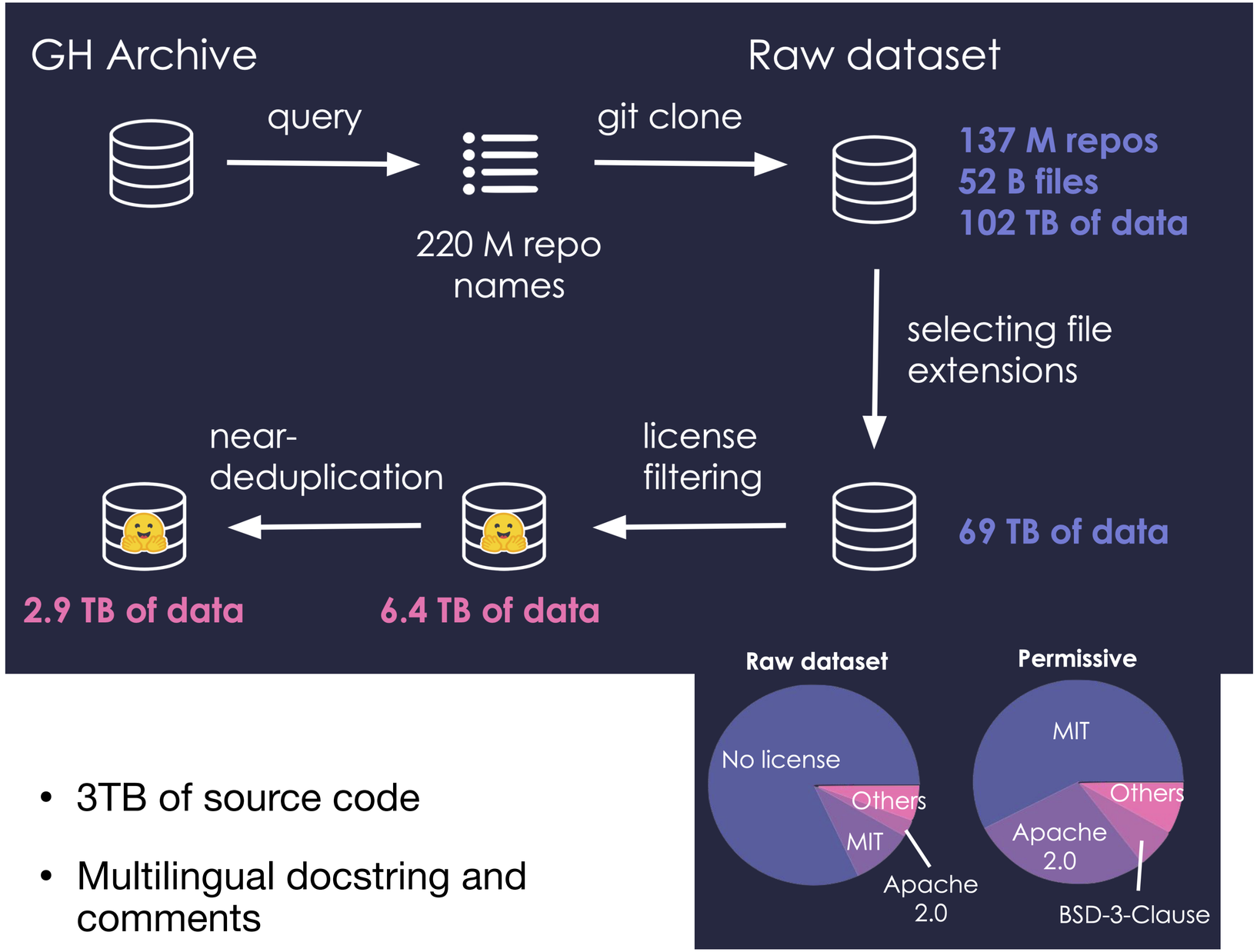


The Common Pile v0.1: An 8TB Dataset of Public Domain and Openly Licensed Text [Kandpal et al. 2025]

The Stack

Code data





- 3TB of source code
- Multilingual docstring and comments

The Stack: 3 TB of permissively licensed source code [Kocetkov et al, ServiceNow / HuggingFace, 2022]

Olmo open model

Open weights, data, checkpoints

- Open-weight models
  - Release final checkpoints
- Open-flow models
  - Release intermediate checkpoints

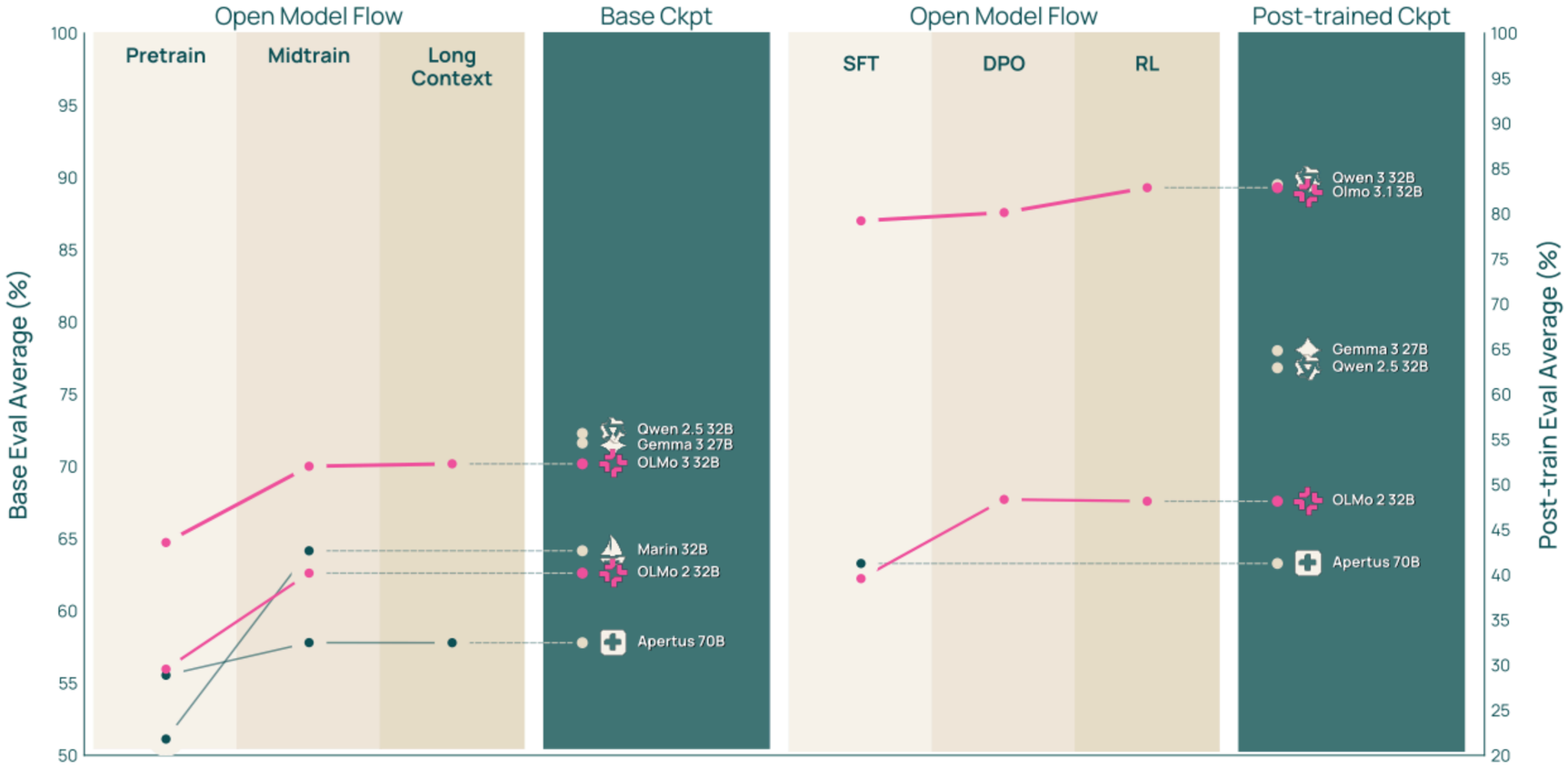


Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Pre-training data

Training on large amount of web data

Source	Type	9T Pool		6T Mix		150B Mix	
		Tokens	Docs	Tokens	Docs	Tokens	Docs
Common Crawl	Web pages	8.14T	9.67B	4.51T (76.1%)	3.15B	121B (76.9%)	84.5M
olmOCR science PDFs	Academic documents	972B	101M	805B (13.6%)	83.8M	19.9B (12.6%)	2.25M
Stack-Edu (Rebalanced)	GitHub code	137B	167M	409B (6.89%)	526M	11.1B (7.06%)	14.3M
arXiv	Papers with LaTeX	21.4B	3.95M	50.8B (0.86%)	9.10M	1.29B (0.82%)	247K
FineMath 3+	Math web pages	34.1B	21.4M	152B (2.56%)	95.5M	4.10B (2.60%)	2.57M
Wikipedia & Wikibooks	Encyclopedic	3.69B	6.67M	2.51B (0.04%)	4.24M	64.6M (0.04%)	119K
Total		**9.31T**	**9.97B**	**5.93T (100%)**	**3.87B**	**157B (100%)**	**104M**

Table 4 Composition of Dolma 3 Mix including our 9T pool of data, the 6T mix we used for final model training, and the 150B mix we used for experimentation.

Table from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Pre-training data

Training on large amount of web data

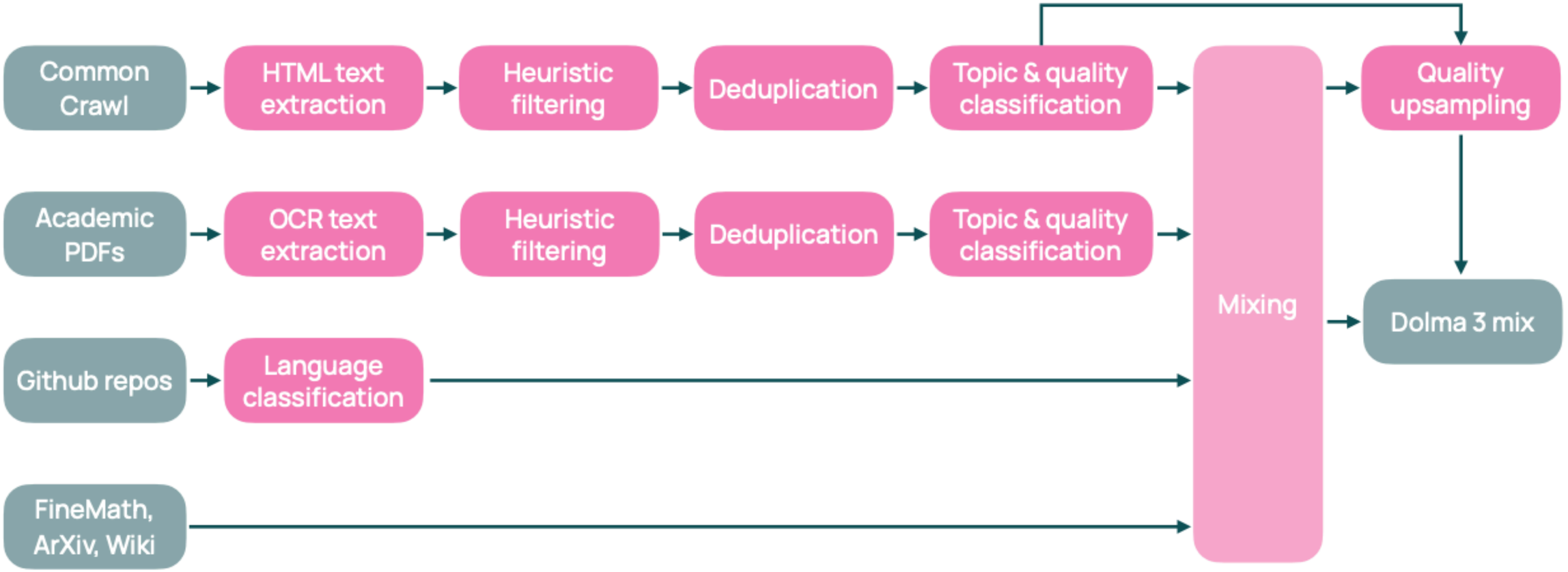


Figure 8 Data curation flow for pretraining data sources in DOLMA 3 MIX.

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Pre-training data

Training on large amount of web data

Pipeline Step	Docs Removed (B)	% of pool removed	% of total time
URL Filters	2.3	0.9	1.68
Length Filters	103.4	40.42	8.03
Symbol Filters	56.5	22.1	4.13
Internal Repetition	32.1	12.53	31.41
Line Modifiers	7.1	2.79	10.0
English Filter	6.2	2.44	14.3
MadLad Filters	9.3	3.65	5.87
Quality Classifiers	0.0	0.0	24.58

Table 36 Web data processing step cost and removal breakdown during the heuristic processing steps. We started with 252.6B documents and ended with 38.7B documents for a total removal rate of 84.9%. This procedure took, in aggregate, approximately 1,030 hours on i4i.32xlarge EC2 instances.

Table from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Why so much data cleaning?

Effect of filtering

DCLM - 240T tokens
from CommonCrawl

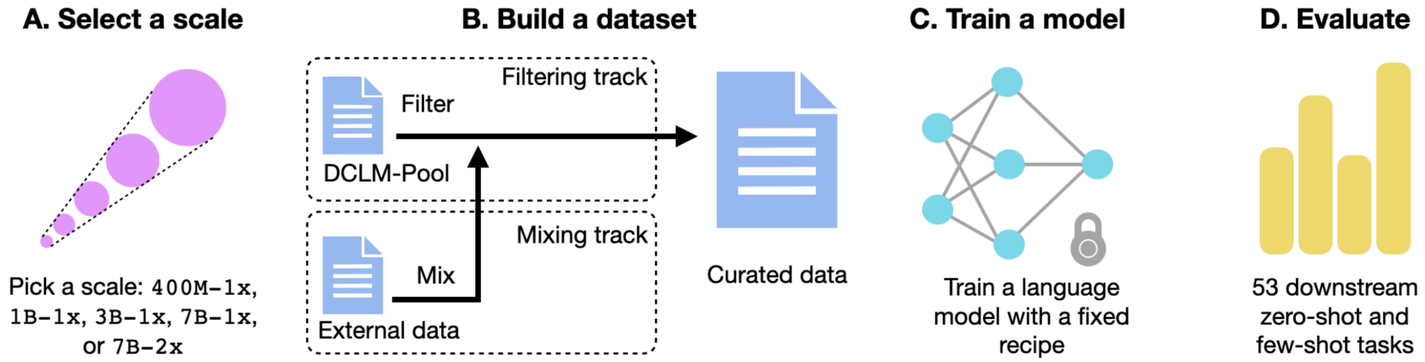


Figure 2: **The DCLM workflow.** *(A)* A participant first chooses a scale, where larger scales reflect more training tokens or model parameters. *(B)* A participant then filters a pool of data (filtering track) or mixes data of their own (mixing track) to create a dataset. *(C)* Using the curated dataset, a participant trains a language model, with standardized training code and scale-specific hyperparameters, which is then *(D)* evaluated on 53 downstream tasks to judge dataset quality.

Filtering matters!

Filter	CORE	EXTENDED
RefinedWeb reproduction	27.5	14.6
Top 20% by Pagerank	26.1	12.9
SemDedup [1]	27.1	13.8
Classifier on BGE features [185]	27.2	14.0
AskLLM [146]	28.6	14.3
Perplexity filtering	29.0	15.0
Top-k average logits	29.2	14.7
`fastText` [87] OH-2.5 +ELI5	**30.2**	**15.4**

fastText ablations (threshold / positives)

Dataset	Threshold	CORE	MMLU	EXTENDED
OH-2.5 + ELI5	10%	**41.0**	**29.2**	21.4
Wikipedia	10%	35.7	27.0	19.1
OpenWebText2	10%	34.7	25.0	18.7
GPT-3 Approx	10%	37.5	24.4	20.0
OH-2.5 + ELI5	15%	39.8	27.2	**21.5**
OH-2.5 + ELI5	20%	38.7	24.2	20.3

DataComp-LM: In search of the next generation of training sets for language models [Li et al, 2025]

Why so much data cleaning?

Effect of filtering

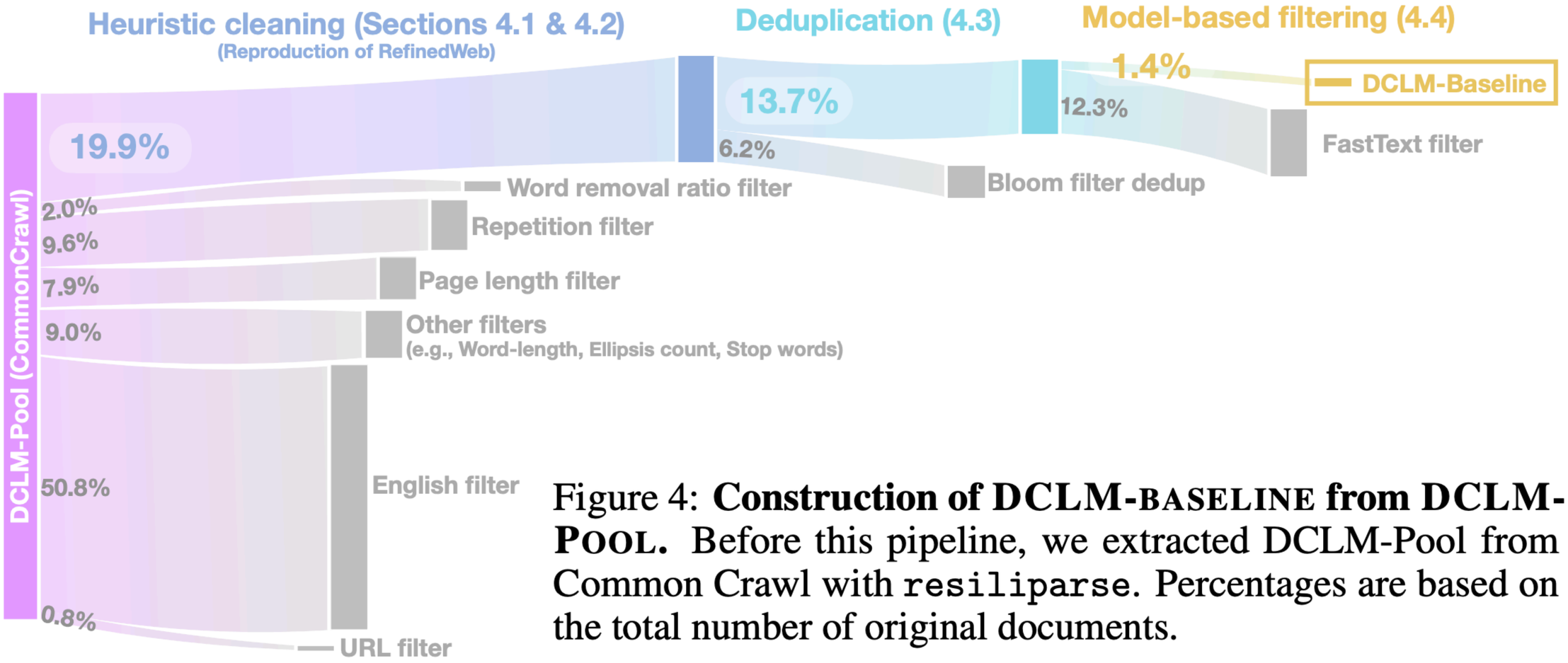


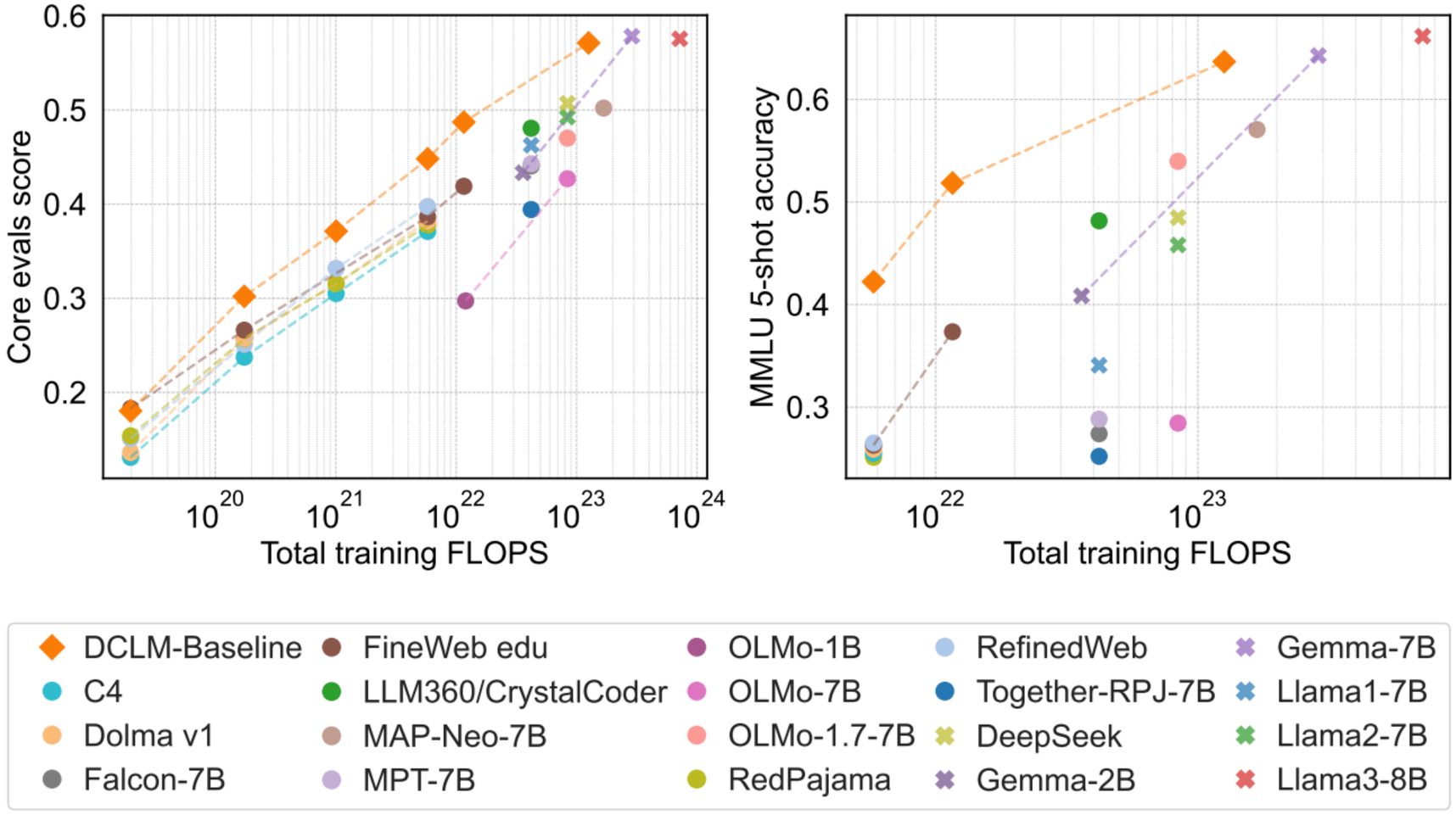
Figure 4: **Construction of DCLM-BASELINE from DCLM-POOL.** Before this pipeline, we extracted DCLM-Pool from Common Crawl with `resiliparse`. Percentages are based on the total number of original documents.

DataComp-LM: In search of the next generation of training sets for language models [Li et al, 2025]

Why so much data cleaning?

Effect of filtering

DCLM - 240T tokens
from CommonCrawl



DataComp-LM: In search of the next generation of training sets for language models [Li et al, 2025]

Why so much data cleaning?

Effect of de-duplicating

By deduplicating

- Cleaner data leads to more efficient training
- Lower perplexity at less tokens processed

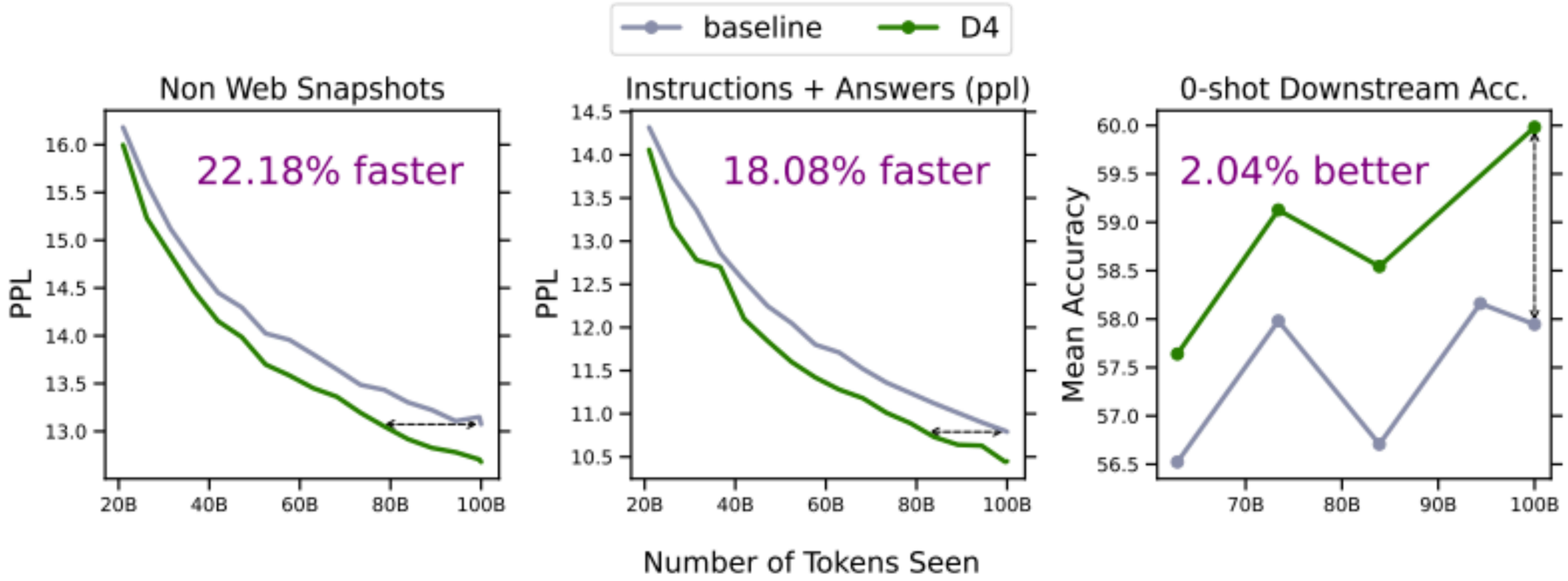


Figure 1: Learning curves for 6.7B OPT model pretraining on 100B tokens, with data selected with D4 (pink line) and randomly (gray line). D4 significantly outperforms baseline training, getting between 18-20% efficiency gains on validation perplexity and 2% increase in average 0-shot downstream accuracy across 16 NLP tasks. See Section A.2 for full learning curves.

D4: Improving LLM Pretraining via Document De-Duplication and Diversification [Tirumula et al., 2023]

Why so much data cleaning?

Effect of de-duplicating

D4: Combines SemDeDup and SSL prototypes

Start with dataset D

- Use SemDeDup on D to get D’
- Cluster points in D’ using K-means
- Apply SSL prototypes on D’

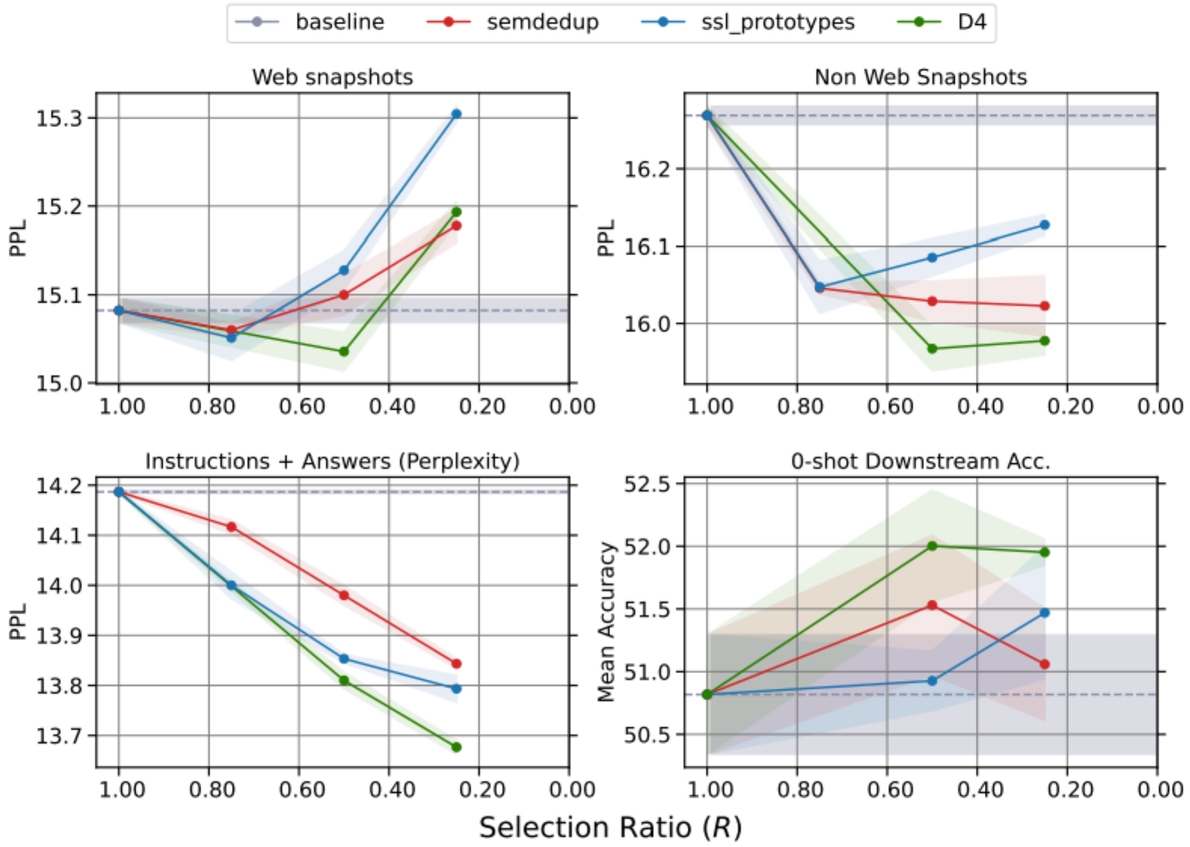


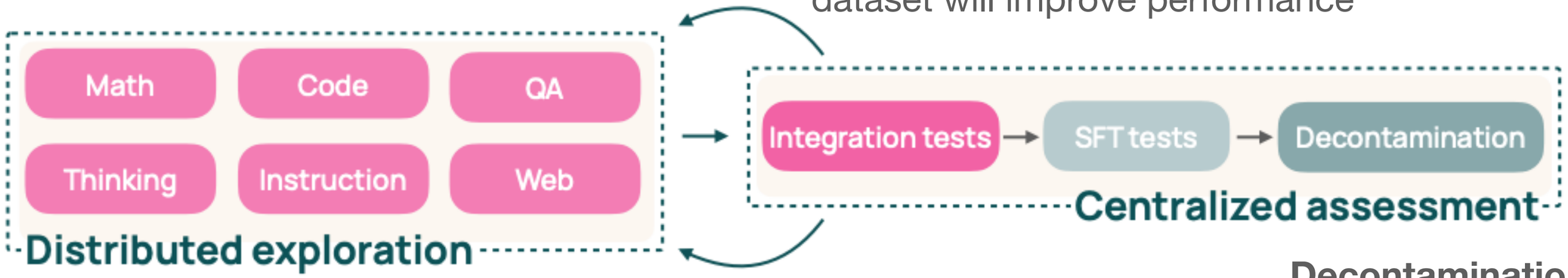
Figure 2: Comparison of data selection methods on validation perplexity. Each point denotes a 1.3B OPT model trained on 40B tokens. The x-axis denotes the selection ratio R. The y-axis for the top 2 and bottom left graph depicts perplexity; the bottom right graph is average downstream on 16 NLP tasks from Zhang et al. [59]. The grey line denotes the value for baseline training. Shaded error is standard error across 3 seeds. **Each point on this graph is trained on the same token budget**: when we decrease R, we jointly increase the size of the source dataset (e.g. choosing 1/4 of documents from a 4x’ed sized source dataset).

D4: Improving LLM Pretraining via Document De-Duplication and Diversification [Tirumula et al., 2023]

Mid-training

Training on cleaned, specialized data

Integration tests (on 100B tokens)
to verify whether including the
dataset will improve performance



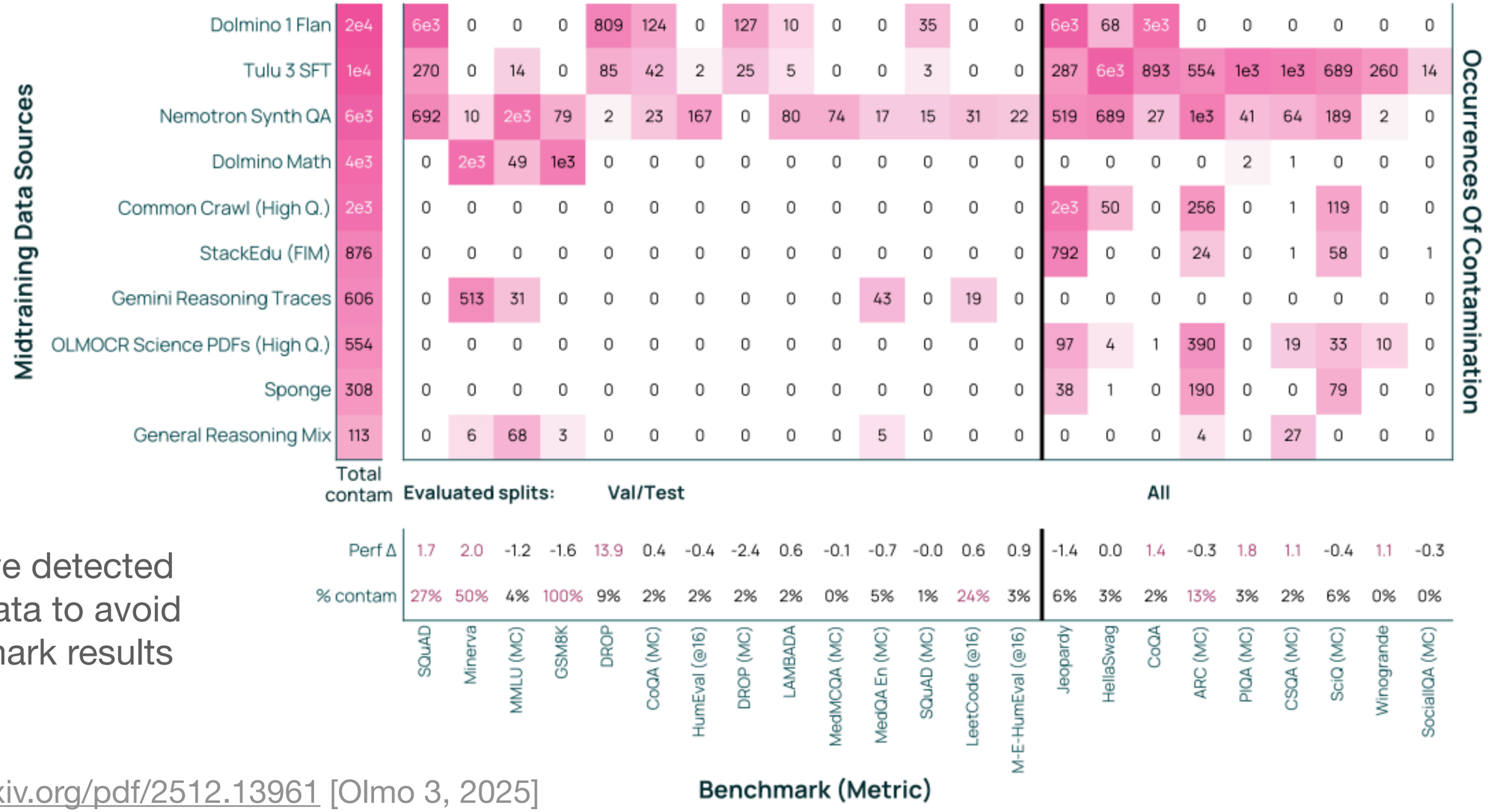
Explore different mixes with small
amounts (5B to 10B tokens)
of sampled data.

Decontamination:
Run detection to try to avoid
benchmark contamination (i.e.
training on evaluation data)

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Mid-training

Datasets with contamination



Fix Mix: Remove detected contaminated data to avoid inflated benchmark results

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Mid-training

Training on long-context data

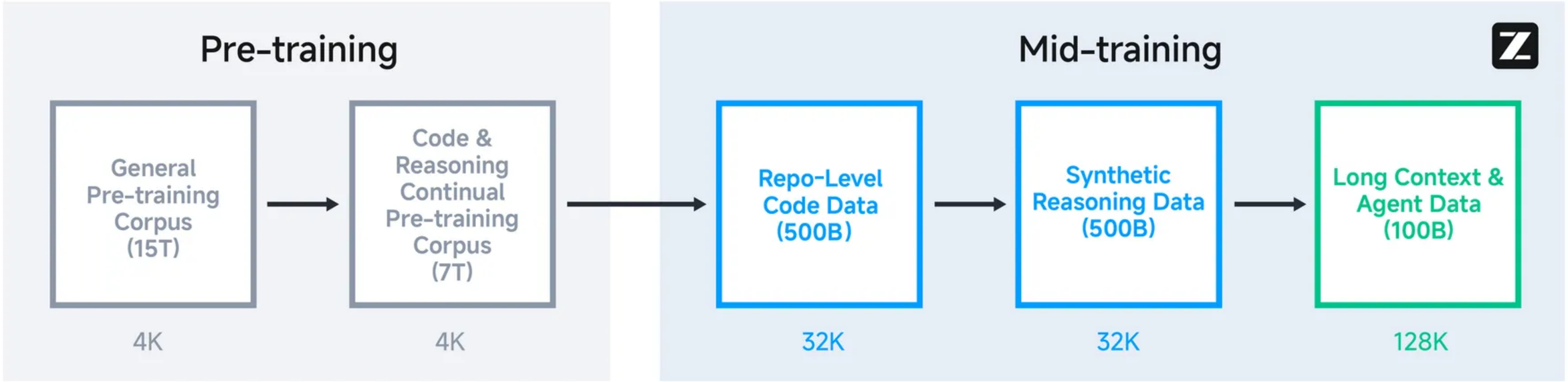
Source	Length bucket	600B Pool		50B Mix	
		Tokens	Docs	Tokens	Docs
olmOCR PDFs	8K-16K	144B (22.5%)	12.7M	2.27B (4.55%)	235K
olmOCR PDFs	16K-32K	115B (18.0%)	5.06M	1.85B (3.70%)	110K
olmOCR PDFs	32K-64K	106B (16.6%)	2.30M	4.81B (9.63%)	177K
olmOCR PDFs	64K-128K	96.0B (15.0%)	1.05M	–	–
olmOCR PDFs	128K-256K	60.8B (9.5%)	342K	–	–
olmOCR PDFs	256K-512K	35.1B (5.49%)	97.1K	–	–
olmOCR PDFs	512K-1M	21.5B (3.36%)	30.2K	–	–
olmOCR PDFs	1M+	26.9B (4.21%)	12.2K	–	–
olmOCR PDFs + synth **CWE**	32K-64K	8.77B (1.37%)	189K	1.94B (3.88%)	71.3K
olmOCR PDFs + synth **REX**	32K-64K	24.1B (3.77%)	492K	6.08B (12.2%)	217K
Midtraining data mix	Variable	–	–	33.0B (66.1%)	79.2M
Total		**639B**	**22.3M**	**50.0B (100%)**	**80.0M**

Table 11 Composition of Dolma 3 Longmino Mix. The 100B mix for OLMO 3 32B maintains the same proportions as the 50B mix. Length buckets are reported in DOLMA 3 tokens.

Table from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Synthetic data for training

- Use of synthetic data in training

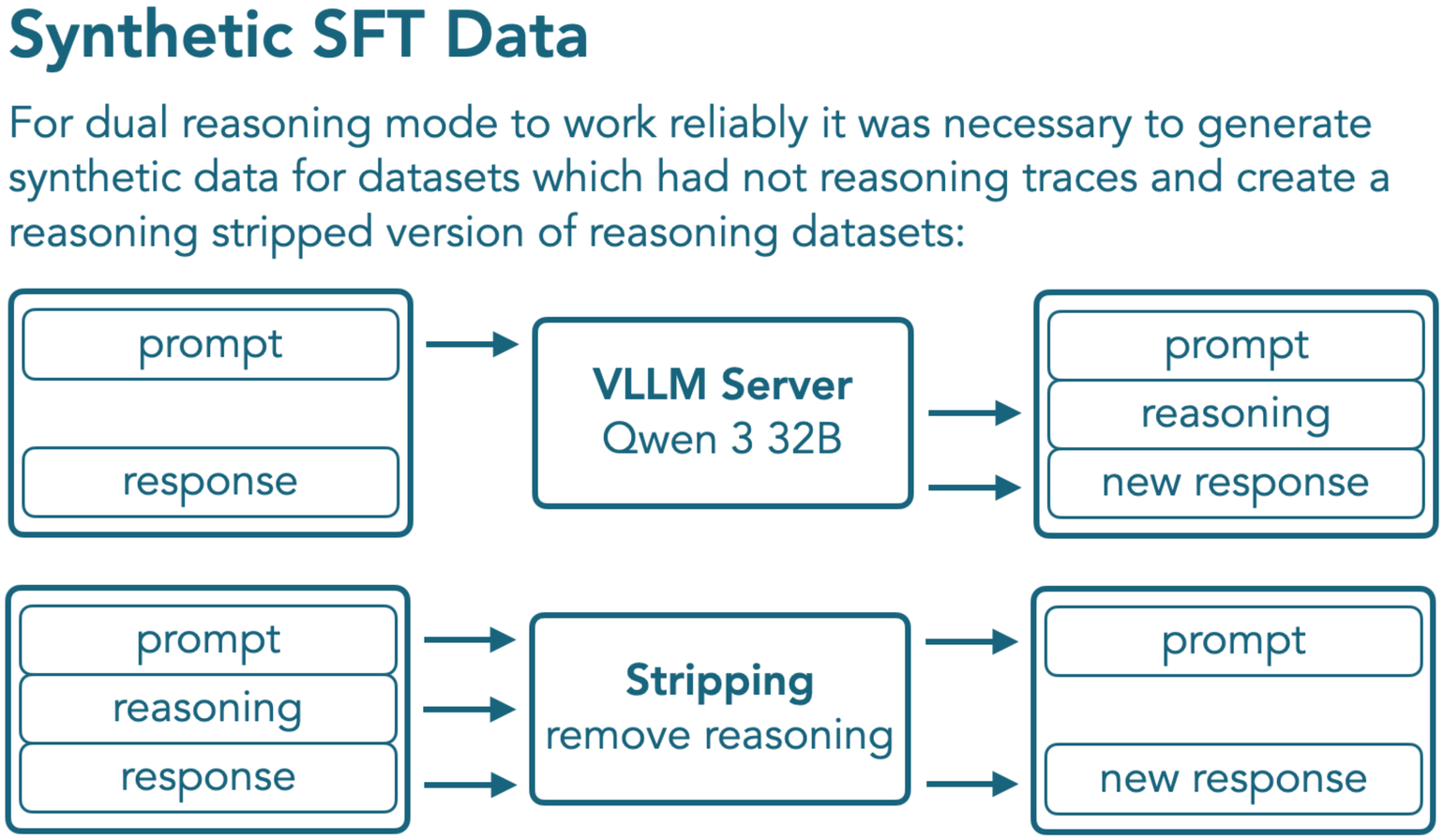


https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- **Data**
- Training process
- Infrastructure

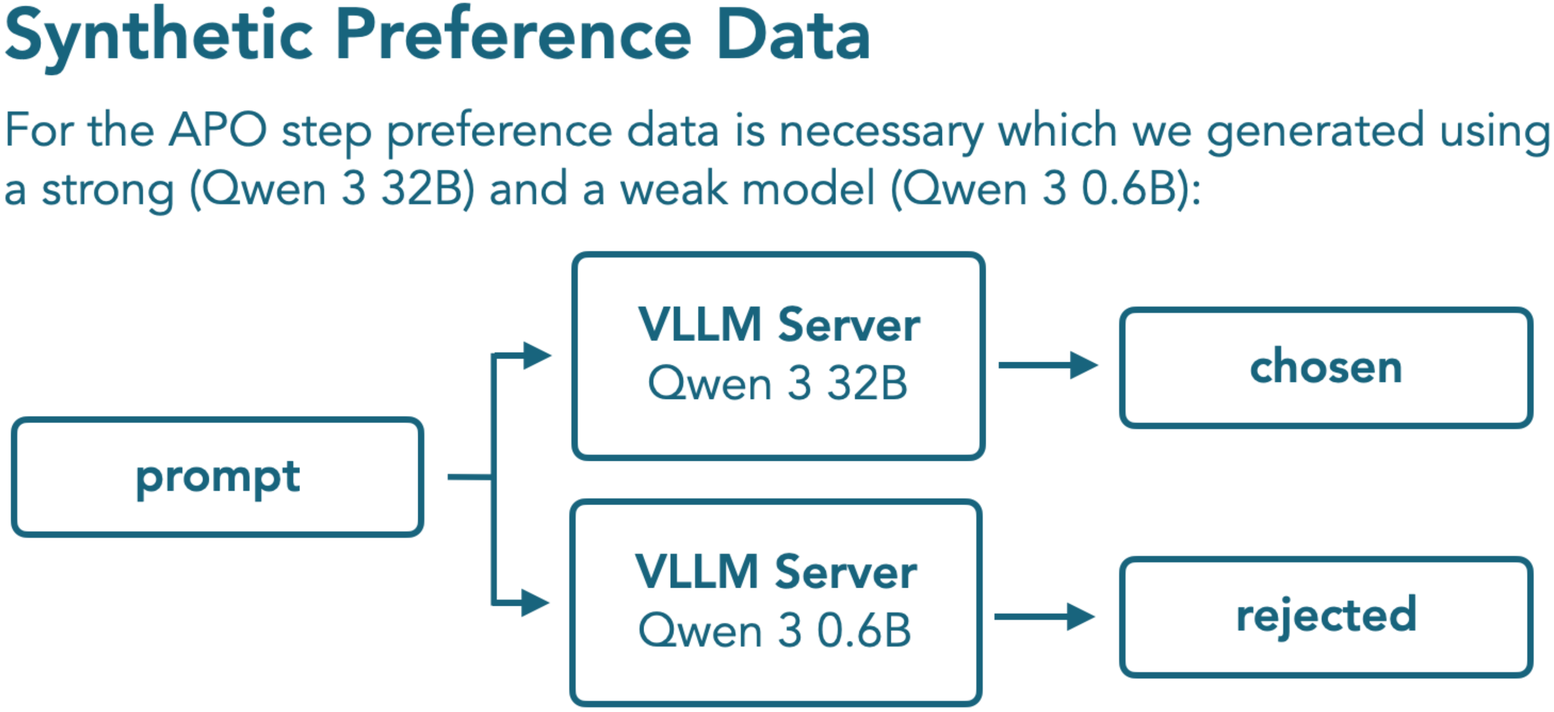


https://huggingface.co/blog/smollm3

HuggingFace guide to pretraining

https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook [HuggingFace, 2025]

- When is it desirable to pretrain?
- Ablations
- Model architecture
- **Data**
- Training process
- Infrastructure



https://huggingface.co/blog/smollm3

Synthetic data for training

- FinePhrase - 486B tokens
- Rephrasing web data
- Investigate how rephrasing strategy, source data, and rephrasing model affects final performance

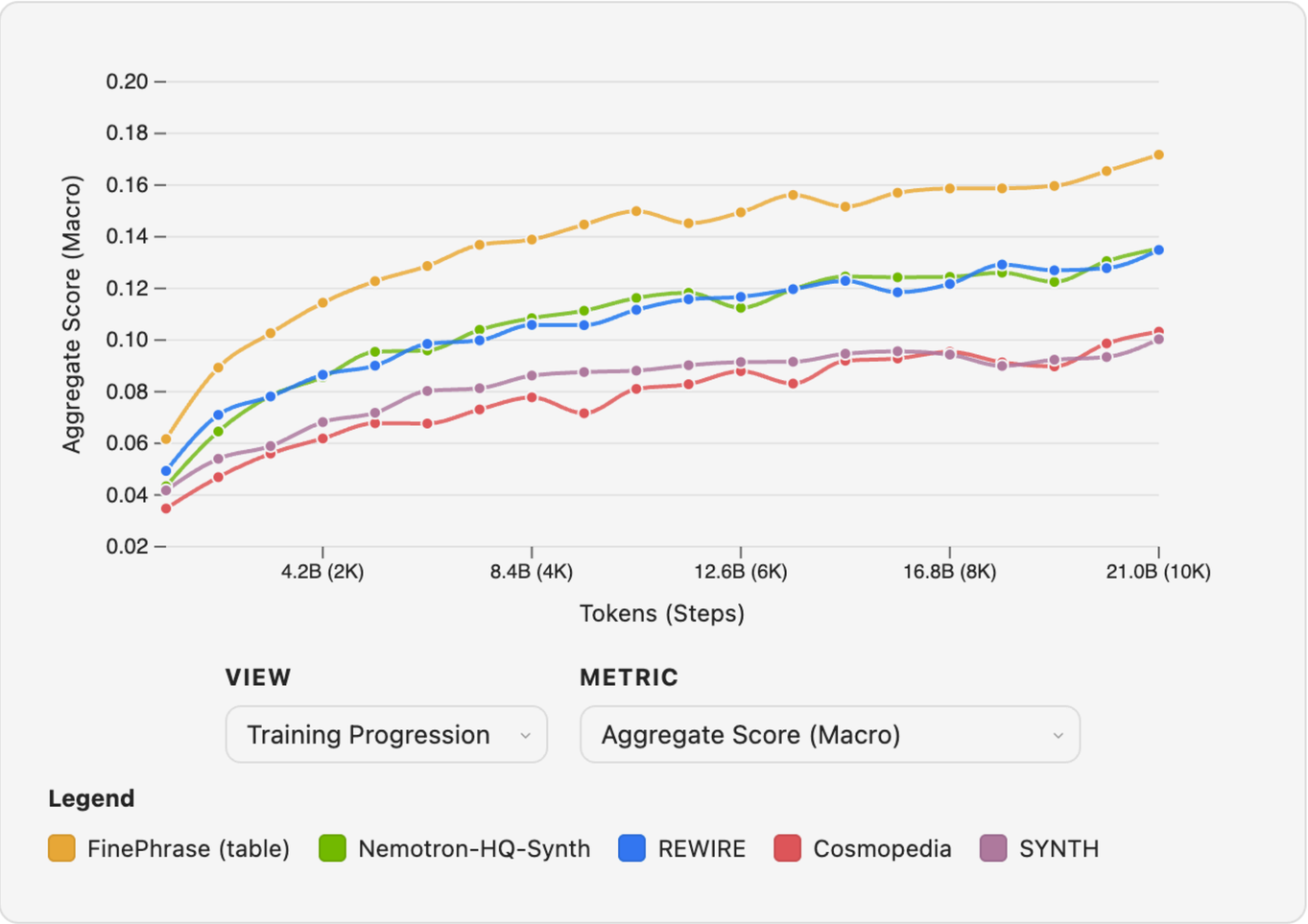
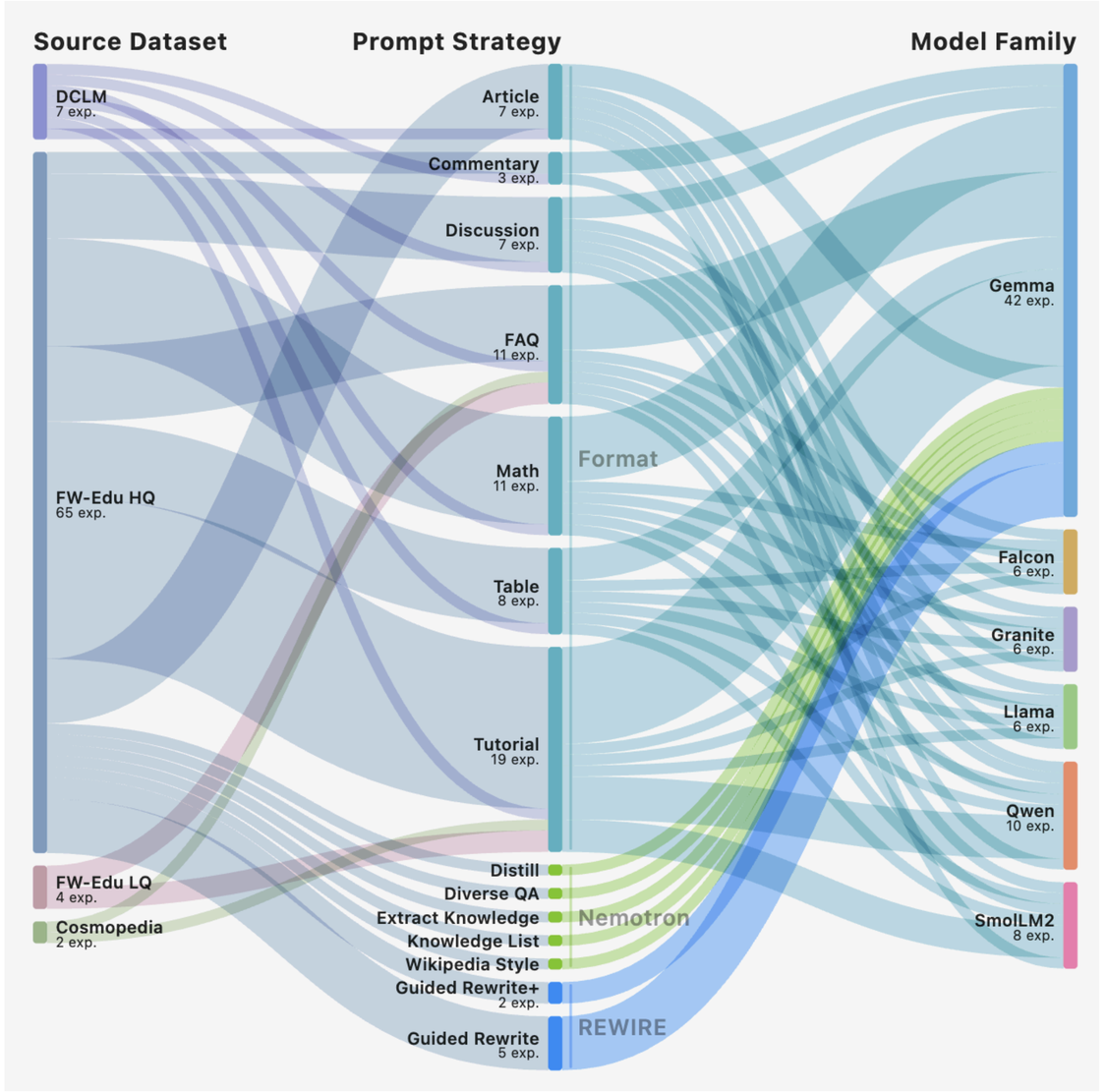


Figure 1: FinePhrase compared against synthetic data baselines across evaluation metrics.

https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

Synthetic data for training

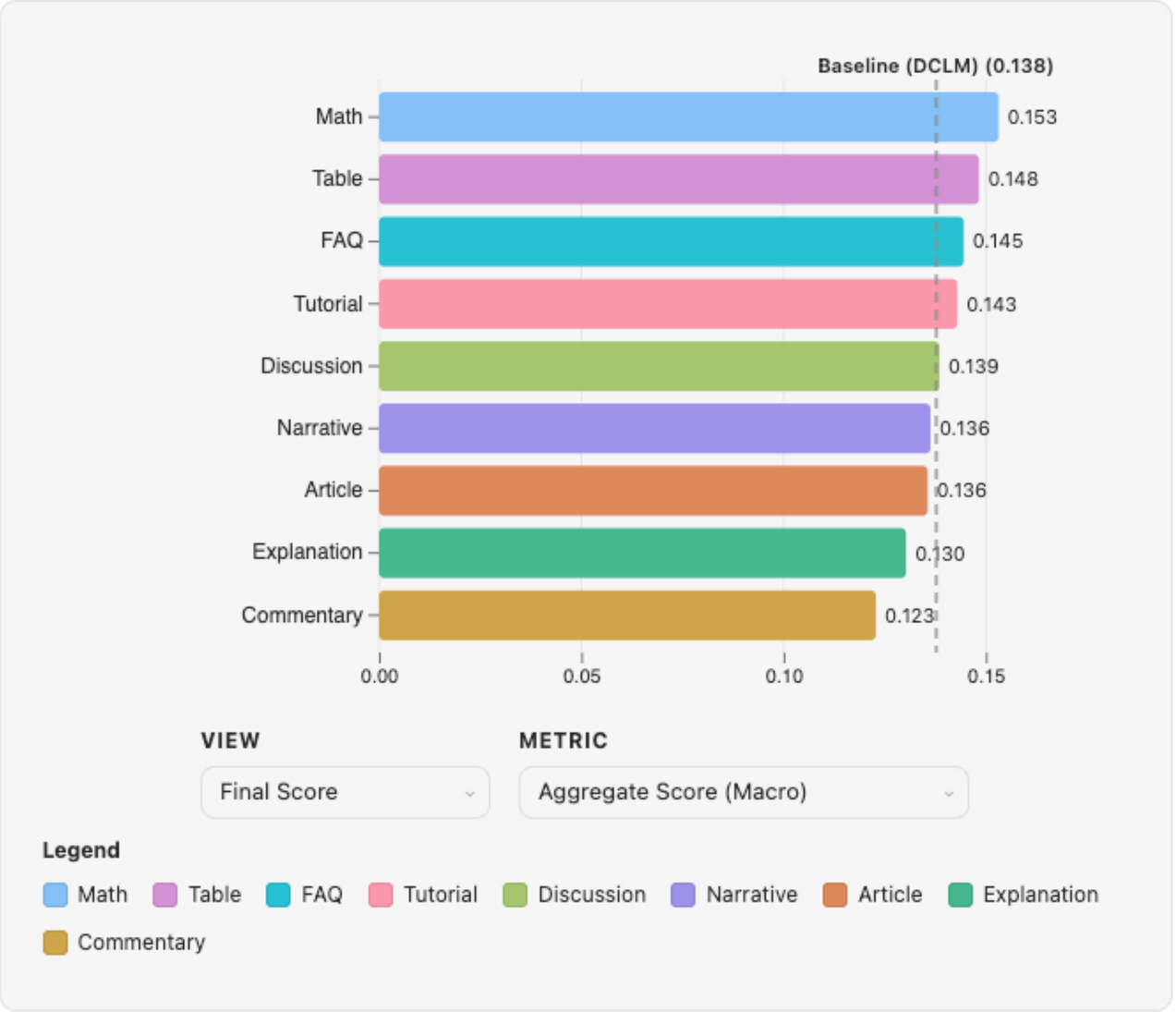
90 Experiments comparing source dataset, prompting, model family



https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

Synthetic data for training

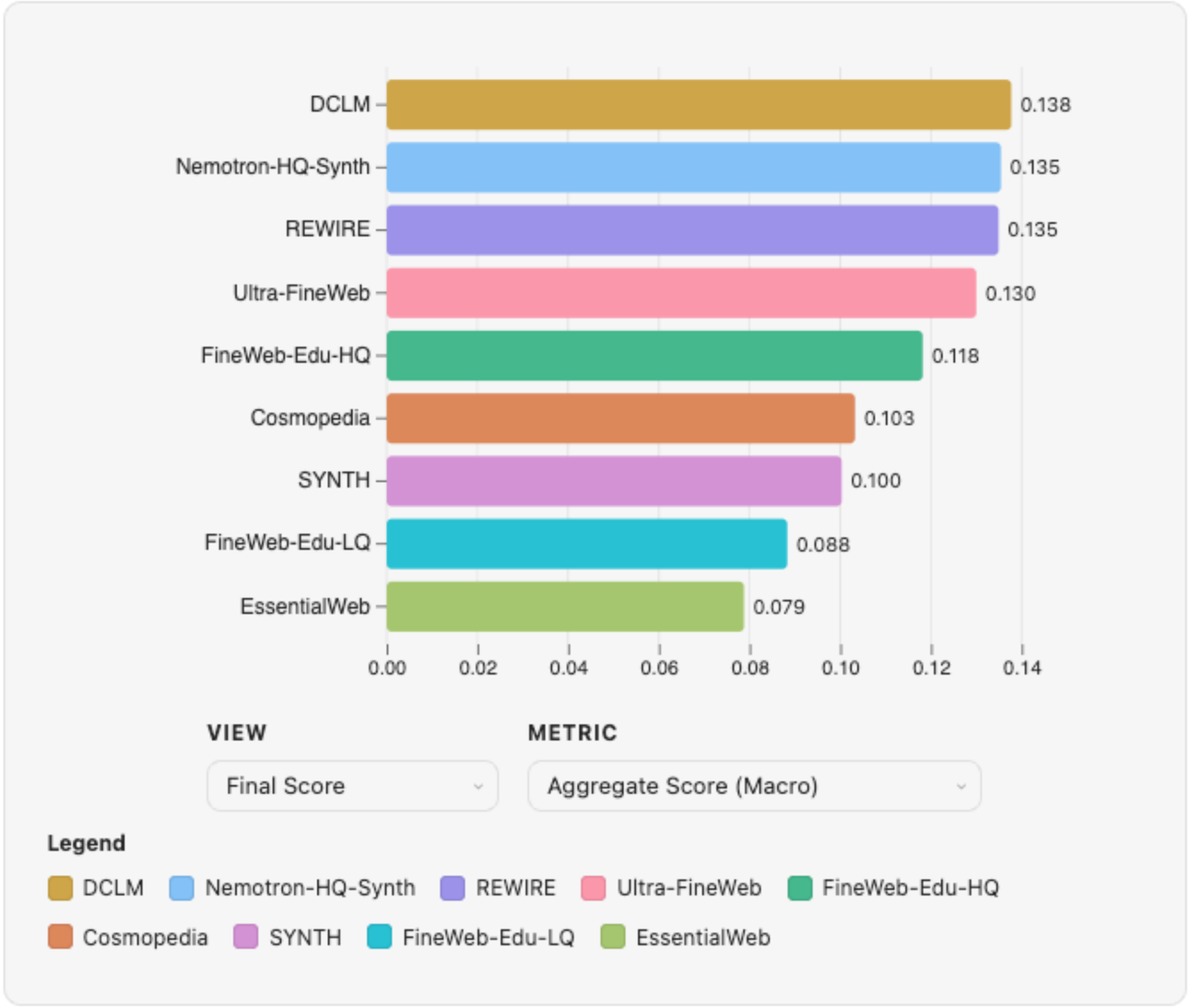
Improving prompting



https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

Synthetic data for training

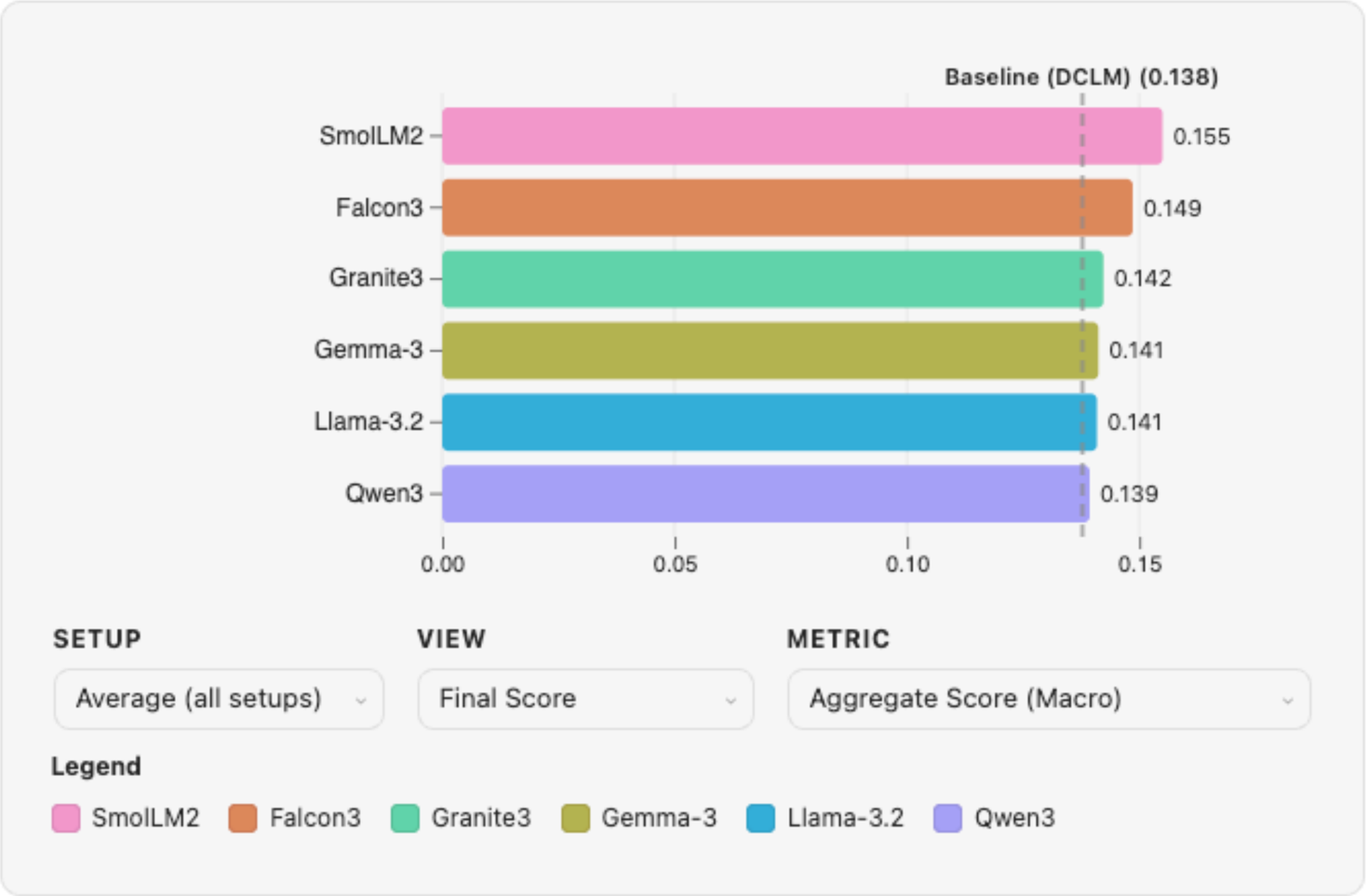
Selecting a source dataset



https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

Synthetic data for training

Selecting a model family



https://huggingface.co/spaces/HuggingFaceFW/finephrase [HuggingFace, 2026]

How does LLM performance scale as we increase model and data size?

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

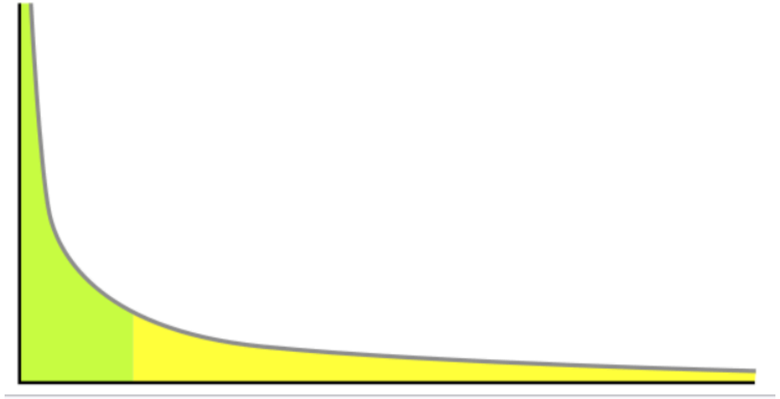
Dario Amodei
OpenAI
damodei@openai.com

https://arxiv.org/abs/2001.08361

Jan 2020

Scaling Laws for LLMs

Power laws



For LLMs, we are interested in how the test performance scales with relation to

- **Model size**: number of model parameters **N** (excluding subword embeddings)
- **Data size**: number of tokens trained on **D**
- Amount of **compute** (MFLOPs) **C** (1 PetaFLOP-day (PF-day) is 8.64×10^{19} FLOPS)

Findings

- Model performance scales as **power law** of model size and data size
- Power law: relation between two quantities where one quantity increases as a power of another
  - $f(x) = (a/x)^k$ e.g. model performance vs. model size
- N, D, C are dominant. Other choices in hyperparameters like width vs. depth are less relevant

https://openai.com/research/ai-and-compute

Model size: computing the number of parameters

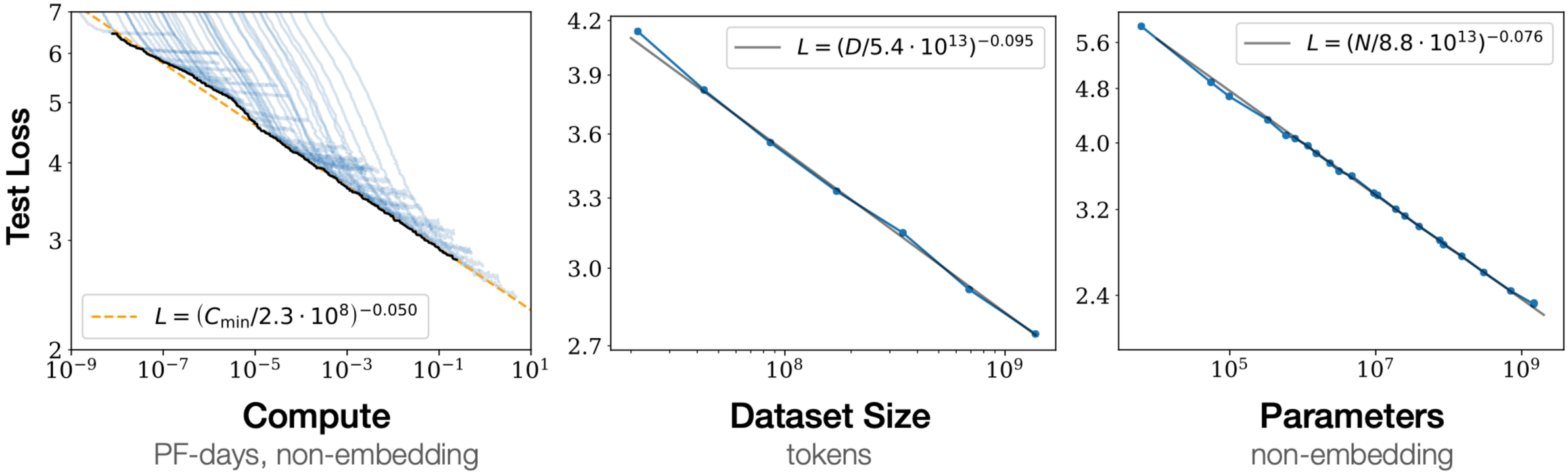
Operation	**Parameters**	**FLOPs per Token**
Embed	$(n_{\text{vocab}} + n_{\text{ctx}})\, d_{\text{model}}$	$4d_{\text{model}}$
Attention: QKV	$n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$	$2n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$
Attention: Mask	—	$2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$
Attention: Project	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2n_{\text{layer}} d_{\text{attn}} d_{\text{embd}}$
Feedforward	$n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$	$2n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$
De-embed	—	$2d_{\text{model}} n_{\text{vocab}}$
Total (Non-Embedding)	$N = 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}})$	$C_{\text{forward}} = 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$

Table 1 Parameter counts and compute (forward pass) estimates for a Transformer model. Sub-leading terms such as nonlinearities, biases, and layer normalization are omitted.

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Test performance

- Power law relationship to compute, dataset size, and number of parameters

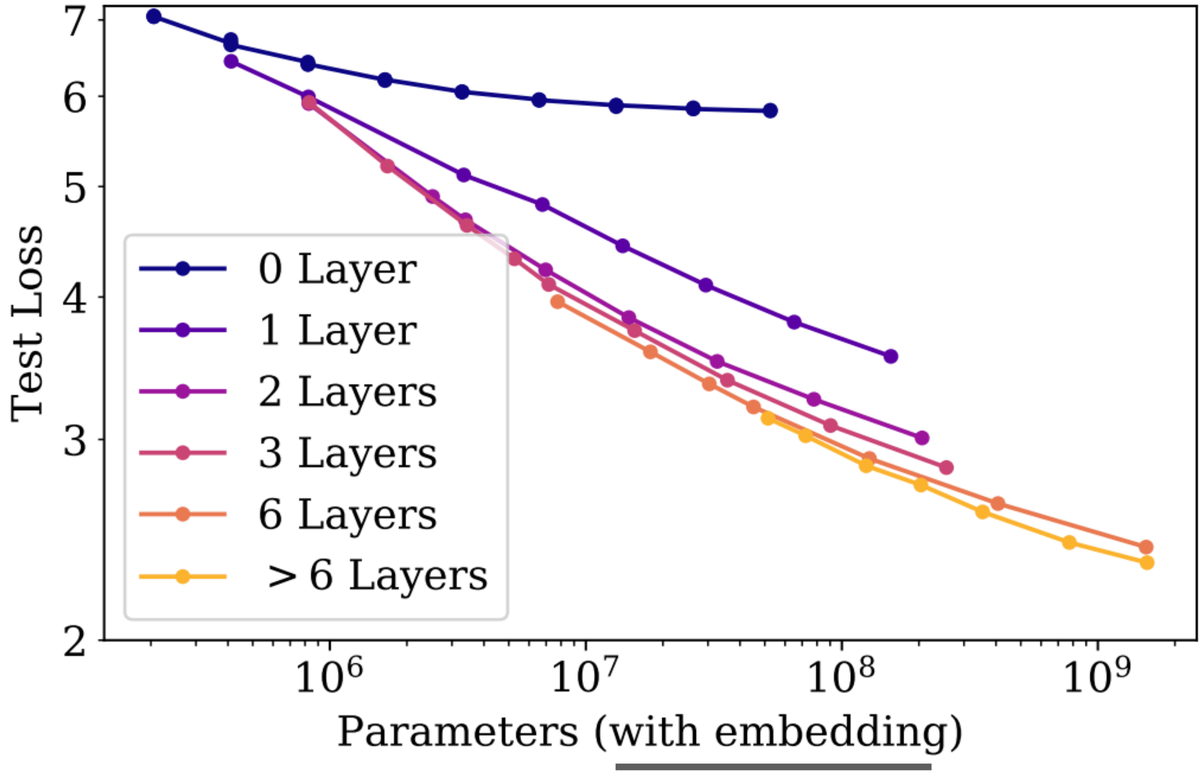


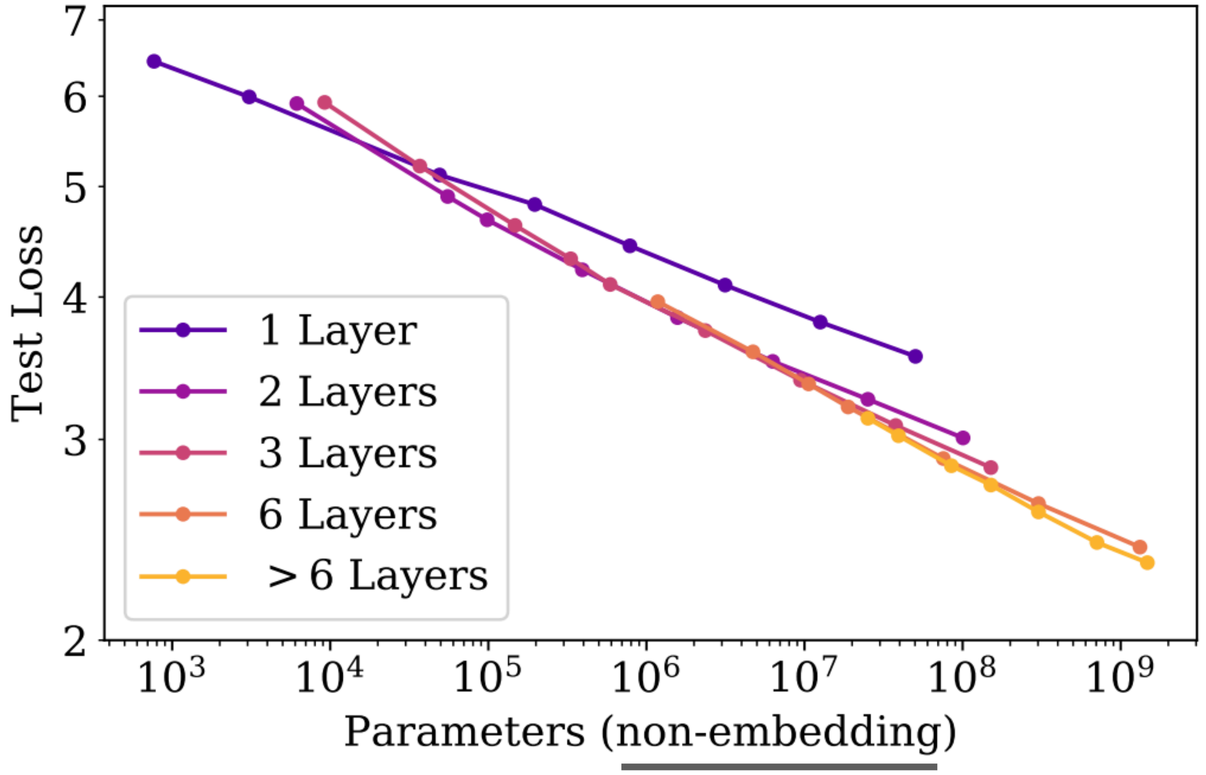
- Language modelling performance increases (test-loss decreases) as compute, dataset size and model size increases
- Need to scale all three in tandem

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Excluding embeddings from parameter count

- Power law relationship not so clear when embeddings are included





Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Power laws for test loss

- Let $L(\cdot)$ represent the test loss dependent on either parameters N, or dataset size D or compute C
- For models with limited number of parameters:

$$L(N) = (N_c/N)^{\alpha_N} \quad \alpha_N \approx 0.076, \quad N_c \approx 8.8 \times 10^{13} \text{ (non-embd params)}$$

- For models with limited dataset size:

$$L(D) = (D_c/D)^{\alpha_D} \quad \alpha_D \approx 0.095, \quad D_c \approx 5.4 \times 10^{13} \text{ (tokens)}$$

- For models trained with limited compute:

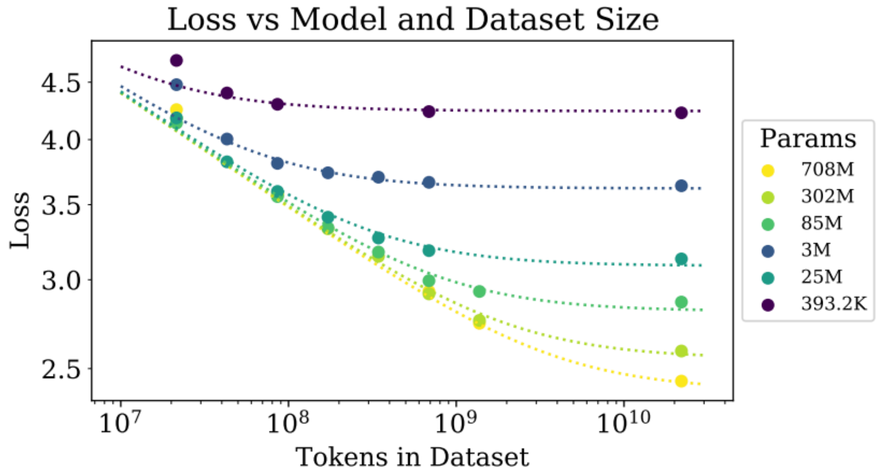
$$L(C) = (C_c^{min}/C_{min})^{\alpha_C^{min}} \quad \alpha_c^{min} \approx 0.050, \quad C_c^{min} \approx 3.1 \times 10^{8} \text{ (PF-days)}$$

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Scaling laws for LLMs

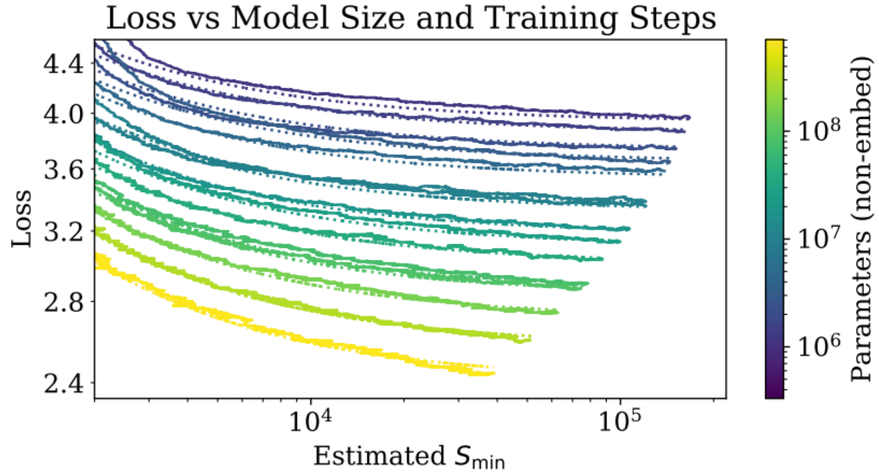
Test loss L as function
of model size N and dataset size D

$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D}$$



Test loss L after transient period as function
of model size N and number of update steps S

$$L(N, S) = \left(\frac{N_c}{N} \right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)} \right)^{\alpha_S}$$

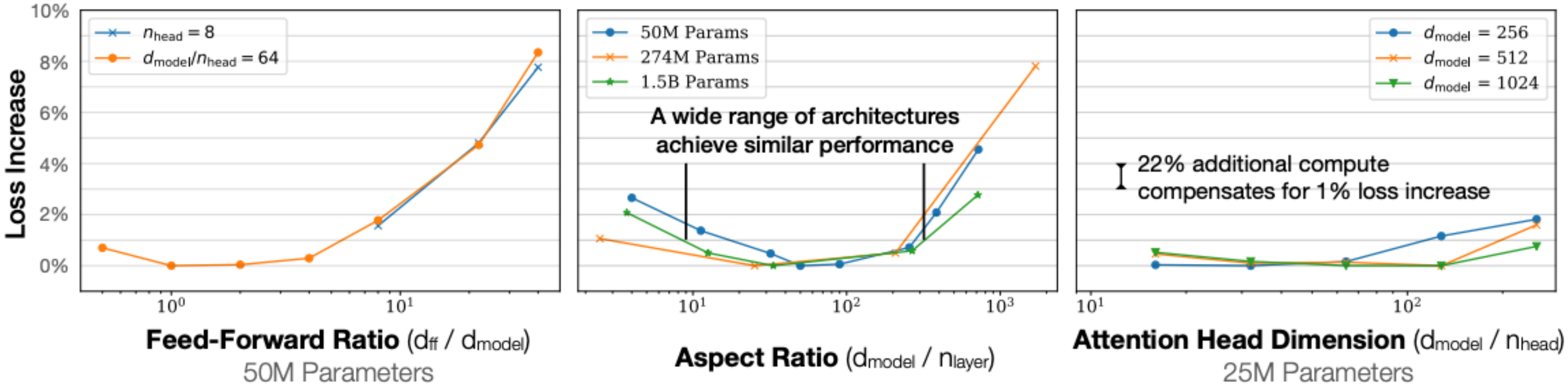


N is number of model parameters (not including vocabulary and positional embeddings)
D is the number of tokens

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Scaling laws for LLMs

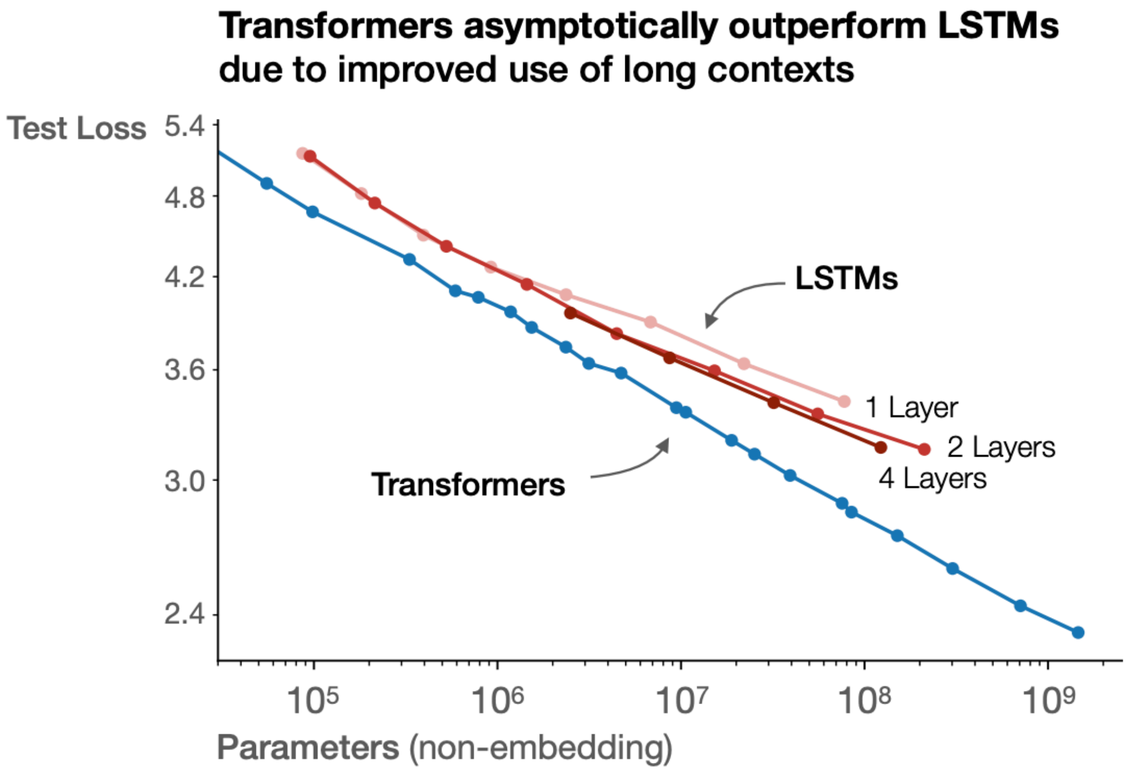
- Keeping model size N fixed, **architecture shape doesn't matter that much**

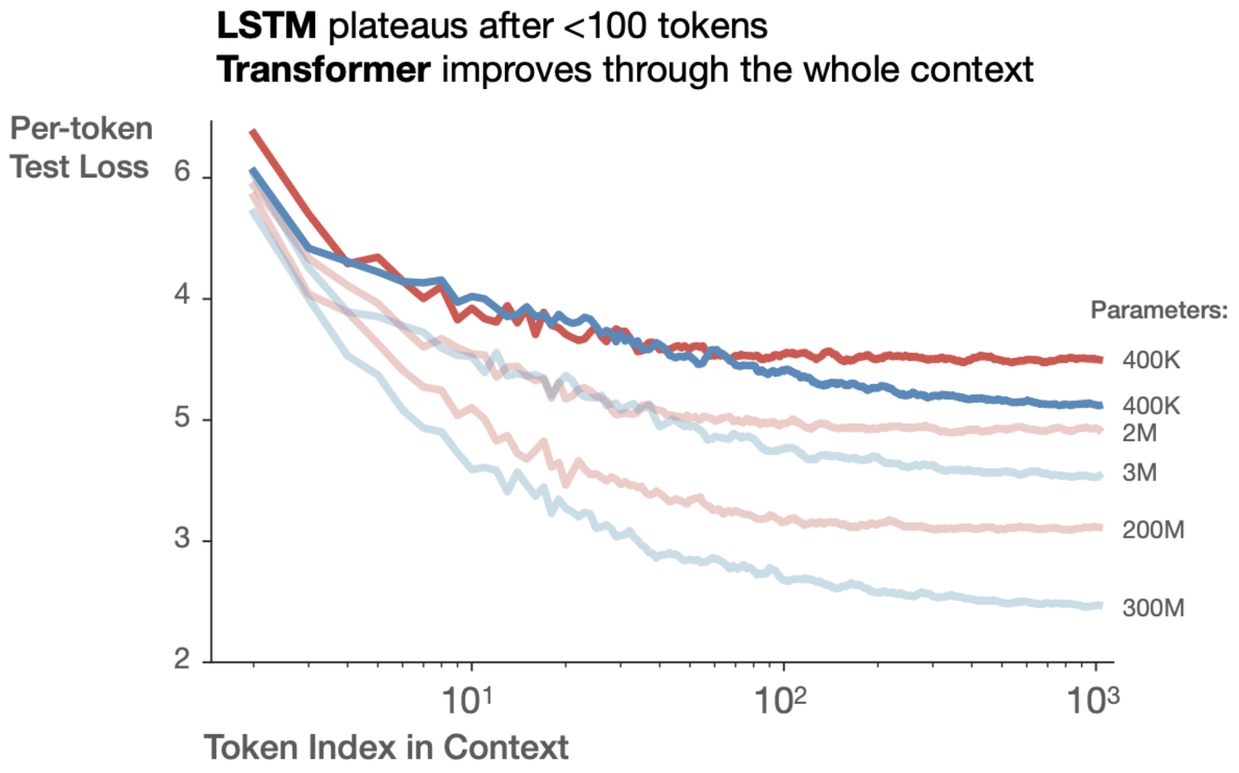


Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Comparing LSTM vs Transformers

- LSTM cannot take advantage of long context (>100 tokens)





Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Given a compute budget, what size model and amount of data should we train on?

Large models are more sample-efficient than small models

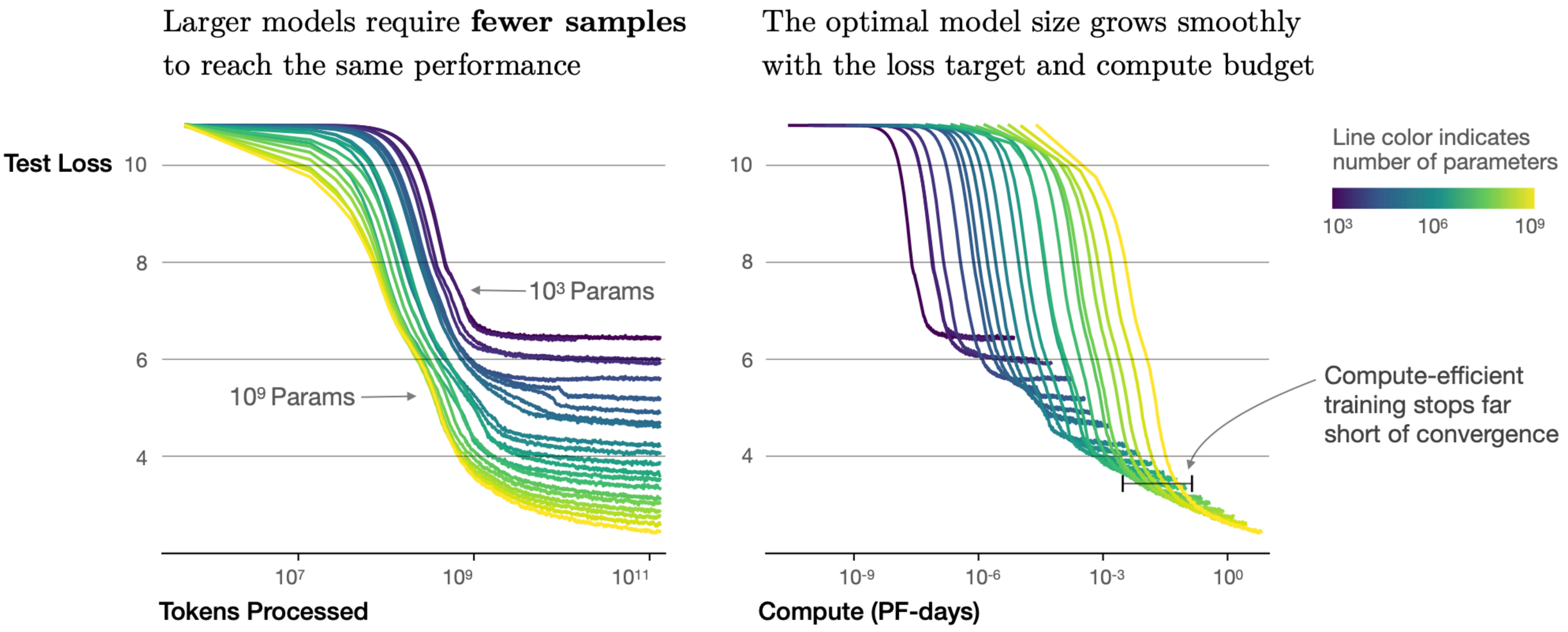


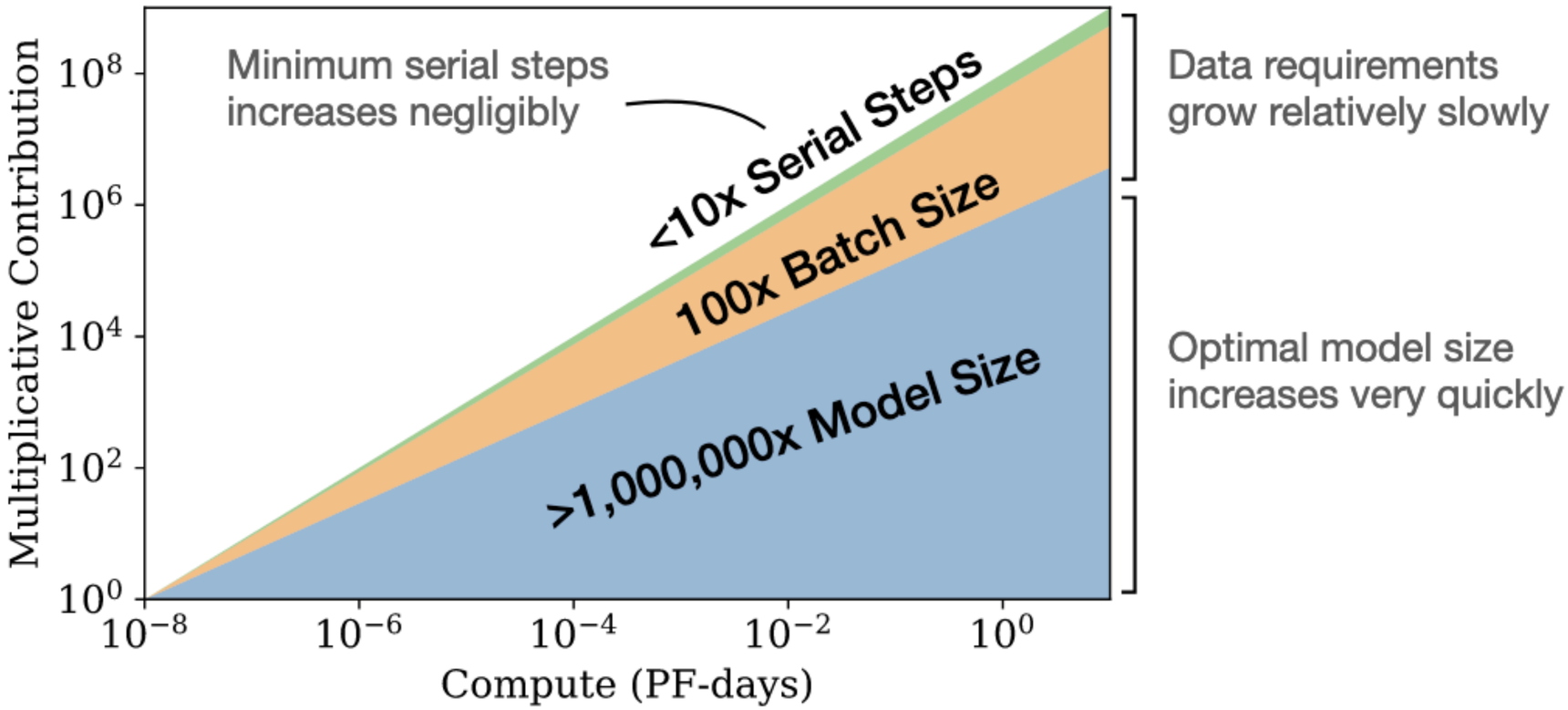
Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

How to allocate increasing compute?

For compute-efficient training

- Increase **model size** more than data (increase data sublinearly).
- Increase **batch size** as data size increases

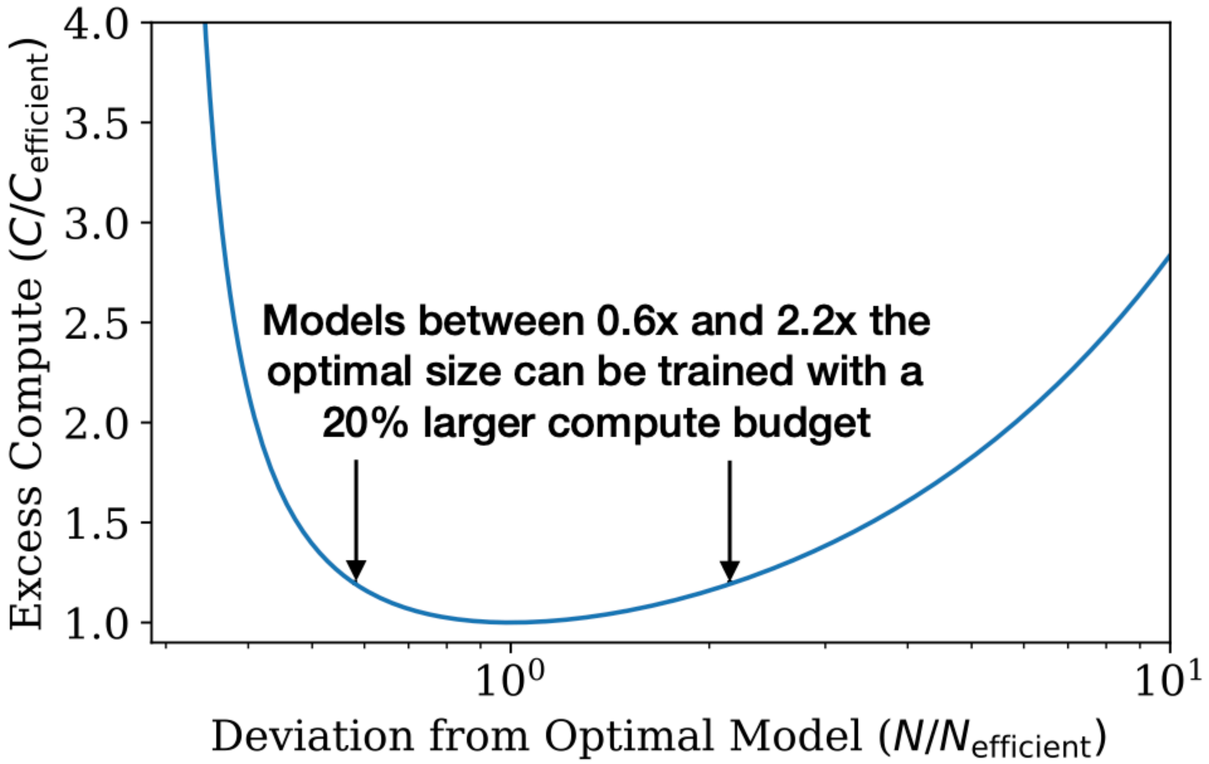


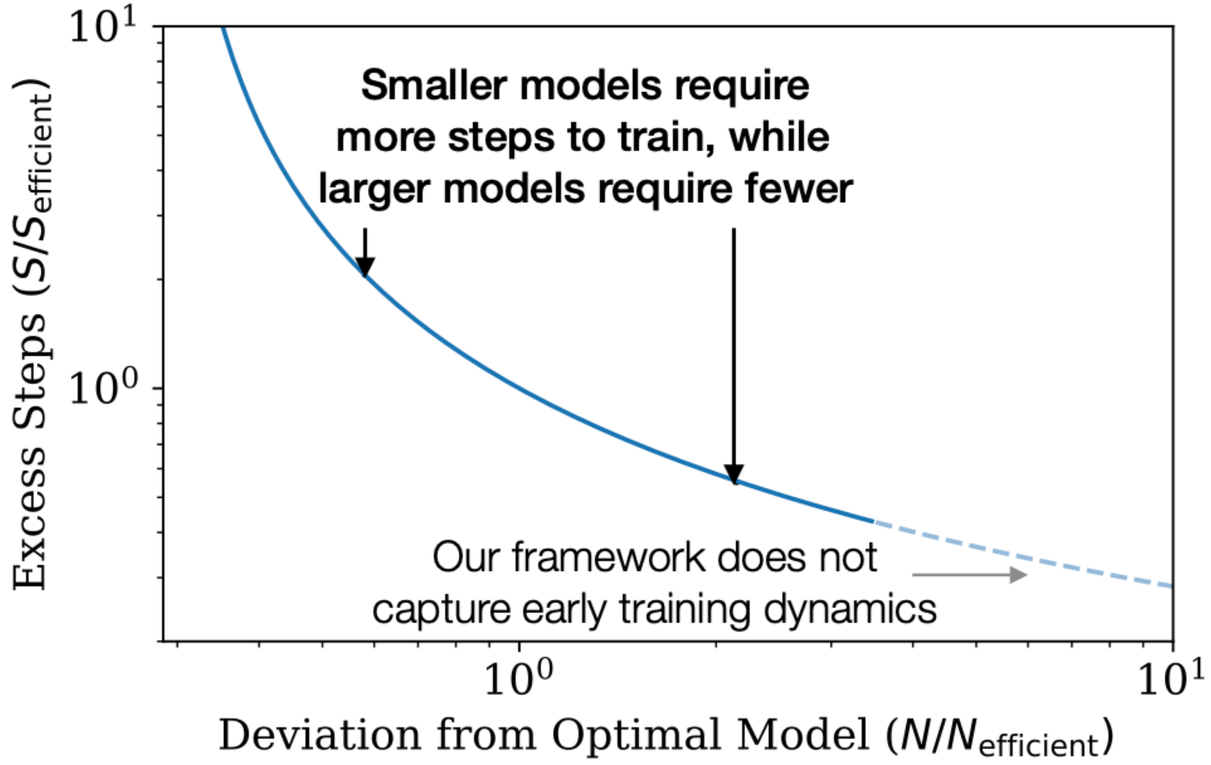
Billion-fold (10^9) increase in compute time

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

Optimal Allocation of Compute Budget

Training at fixed batch size (should increase batch size with more data)





Models larger than the optimal-size can train faster (with less steps)

Scaling laws for neural language models [Kaplan et al. OpenAI, 2020]

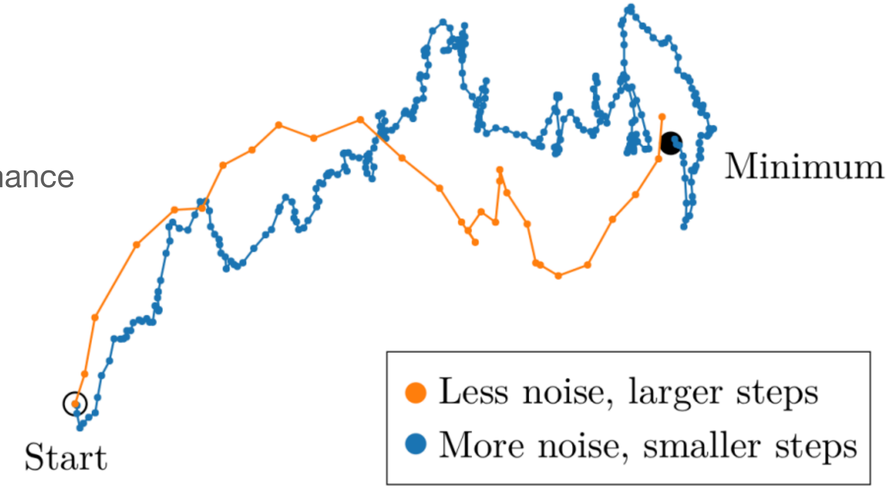
Critical batch size

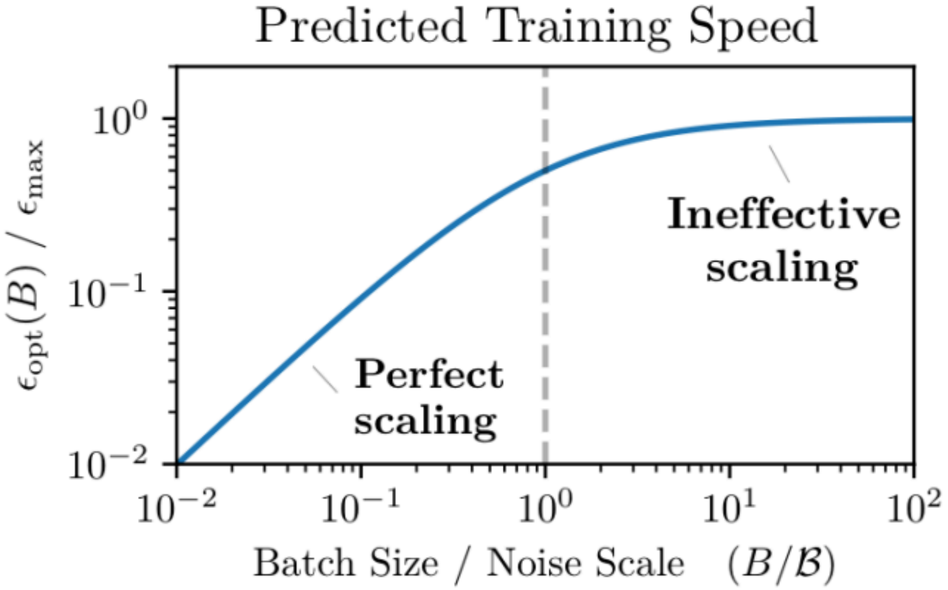
For compute efficient training, train with $B_{\text{crit}} = \frac{E_{\text{min}}}{S_{\text{min}}}$

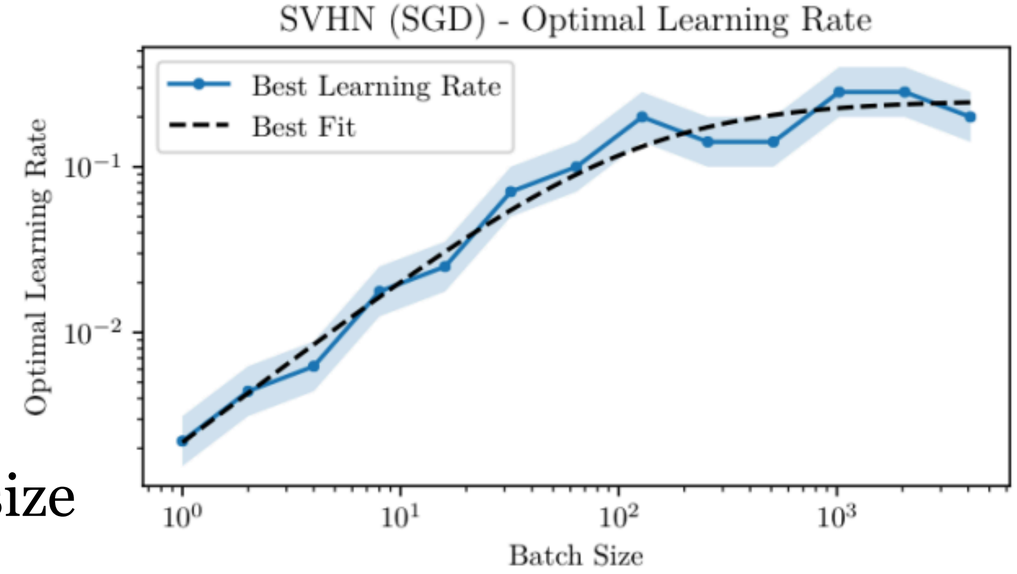
Number of training examples to reach a given performance

Number of training steps

- Larger B: more stable gradient, less training steps
- Critical batch size: above which scaling efficiency decreases significantly







- Optimal learning rate scales linearly with batch size

An Empirical Model of Large-Batch Training [MacCandlish, et al. 2018] - arXiv:1812.061

Critical batch size as function of test loss

$$B_{\text{crit}}(L) \equiv \frac{E_{\text{min}}}{S_{\text{min}}}$$

$$B_{\text{crit}}(L) \approx \frac{B_*}{L^{1/\alpha_B}}$$

$$B_* \sim 2 \cdot 10^8 \text{ tokens}$$

$$\alpha_B \sim 0.21$$

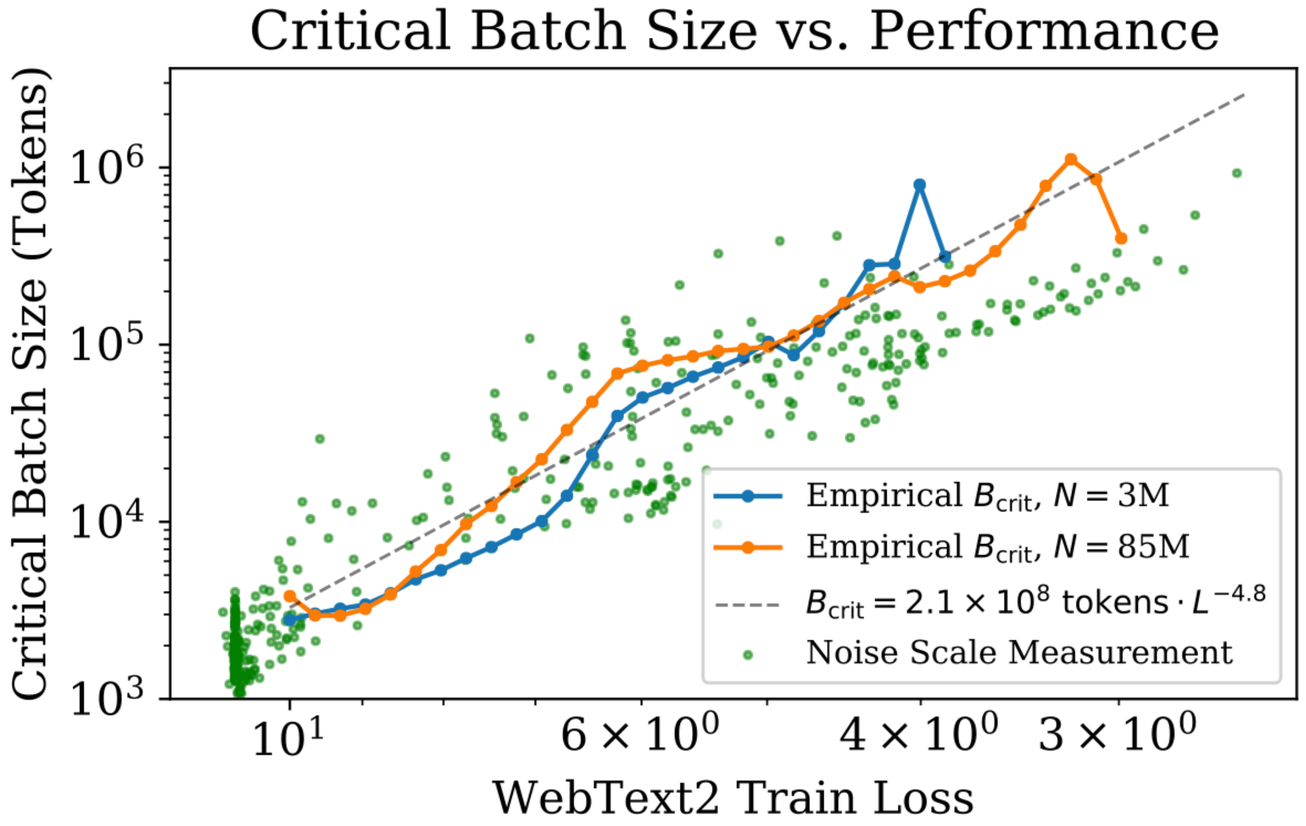


Figure 10 The critical batch size B_{crit} follows a power law in the loss as performance increase, and does not depend directly on the model size. We find that the critical batch size approximately doubles for every 13% decrease in loss. B_{crit} is measured empirically from the data shown in Figure 18, but it is also roughly predicted by the gradient noise scale, as in [MKAT18].

arXiv:1812.06162

Lessons from scaling LLMs

- Number of model parameters
- Size of dataset D
- Amount of compute (MFLOPs) C

- Performance depends **strongly on scale**, **weakly on model shape**
- Performance has a **power-law** relationship with each of the three scale factors N, D, C when not bottlenecked by the other two
- Performance improves predictably as long as we **scale up N and D in tandem**
- Training curves follow predictable power-laws whose parameters are roughly independent of the model size

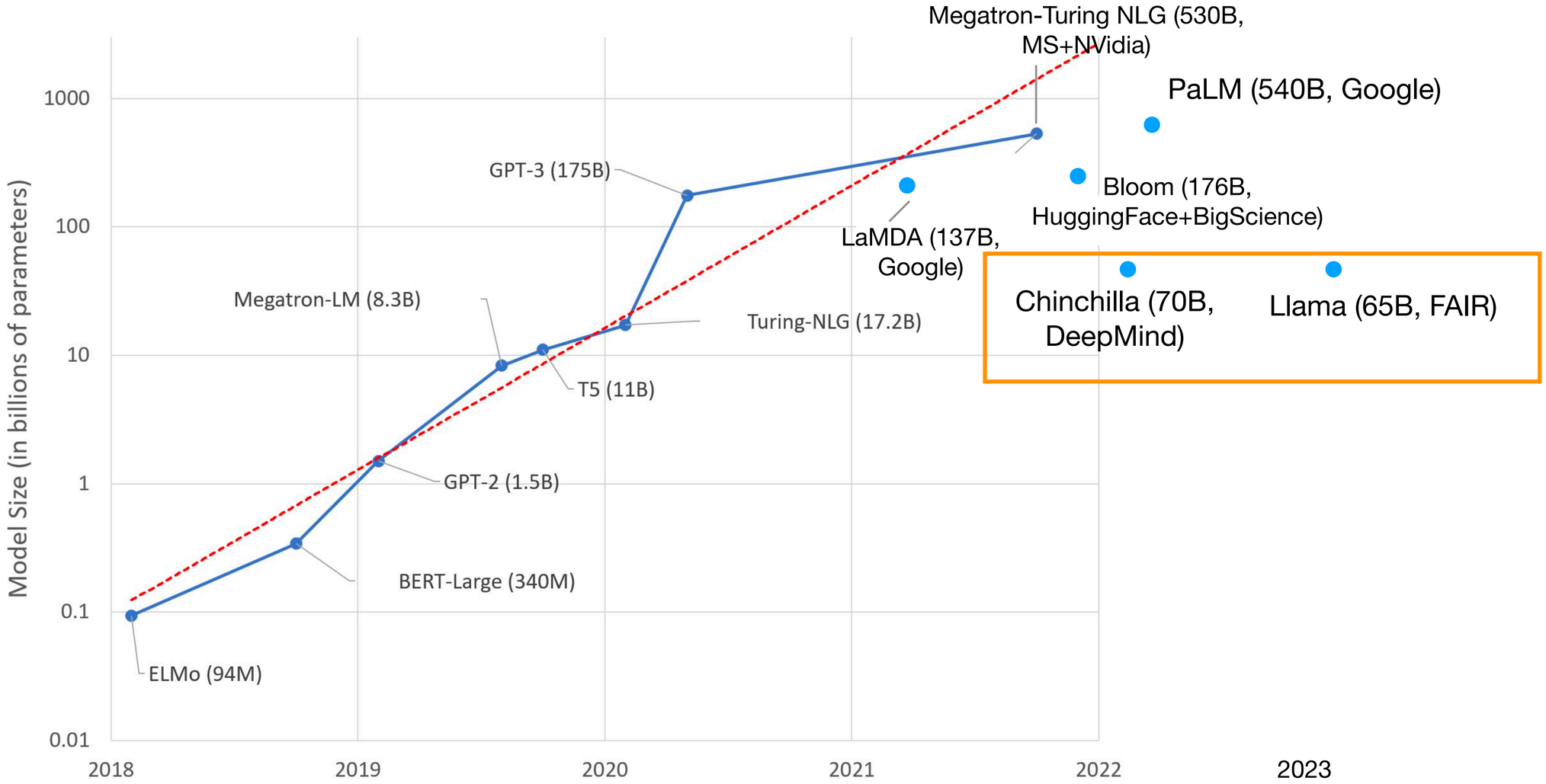
Lessons from scaling LLMs

- Transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set.
- Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps and using fewer data points
- The ideal batch size for training these models is roughly a power of the loss only, and continues to be determinable by measuring the gradient noise scale
- If no constraints on data and model size, with given compute budget C

$$N \propto C^{\alpha_C^{\min}/\alpha_N} \qquad B \propto C^{\alpha_C^{\min}/\alpha_B} \qquad S \propto C^{\alpha_C^{\min}/\alpha_S} \qquad D = B \cdot S$$

Is larger models always better?
Can we train high-performance smaller models with more data?

Is bigger always better?



https://huggingface.co/blog/large-language-models

DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

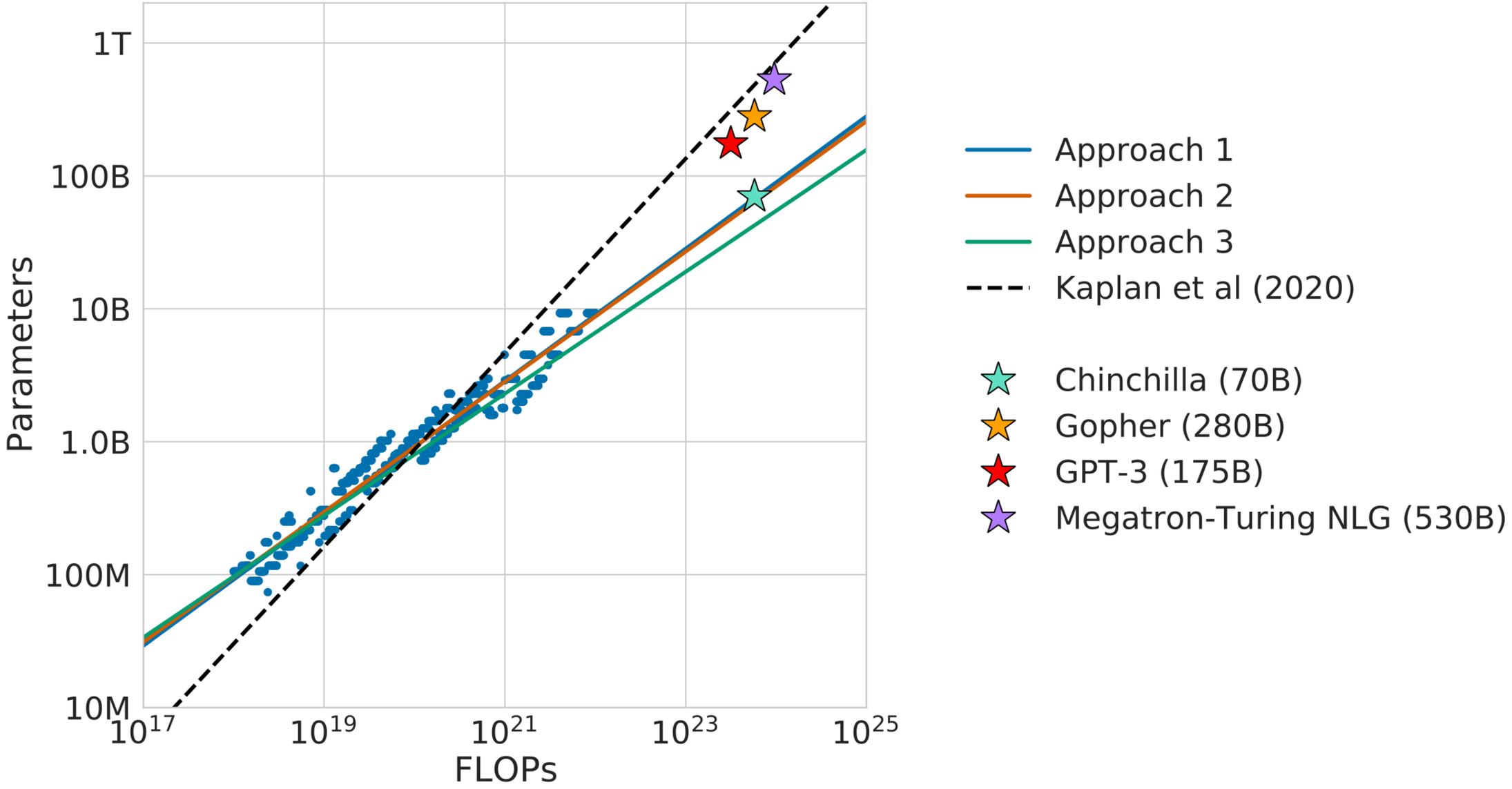
https://arxiv.org/abs/2203.15556

Train longer on more tokens

Lessons from training Chinchilla

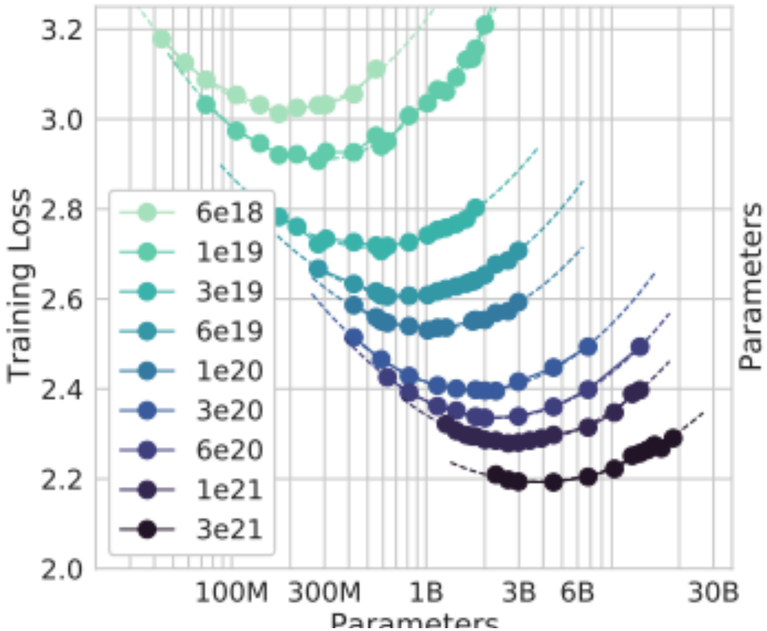
- From GPT3: large models should not be trained to lowest possible loss to be compute optimal
- Question: **Given a fixed FLOPs budget how should one trade off model size and number of training tokens?**
- Pre-training loss L(N, D) for N parameters and D training tokens. Find the optimal N and D values for a given compute budget.
- Empirical study on training 400 models from 70M to 16B parameters, trained on 5B to 400B tokens.
- Answer: **Train smaller models for (a lot) more training steps**.

- **Better to scale model size and number of tokens linearly!**

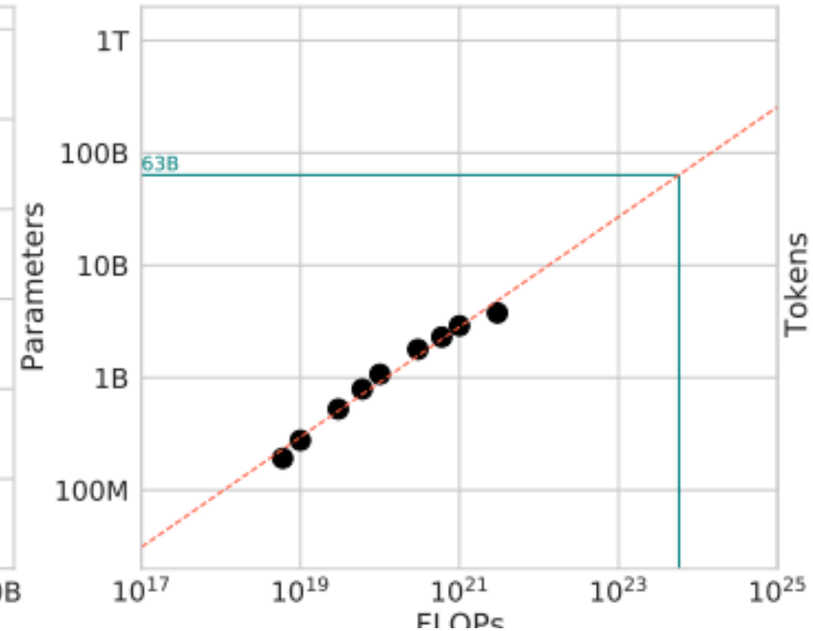


- For different model sizes, choose number of training tokens to keep FLOPs constant

IsoFLOP



Model size



Data size

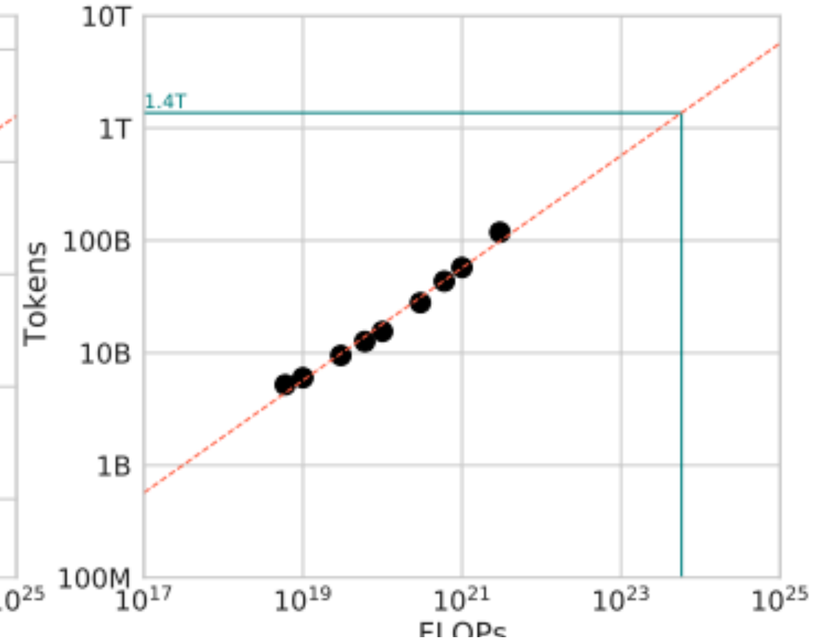
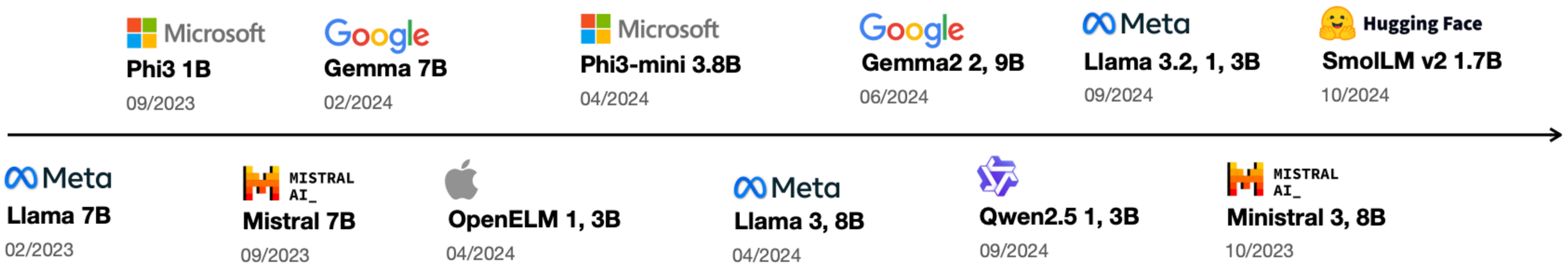


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Is larger models always better?
Can we go to even smaller models?

Small language models

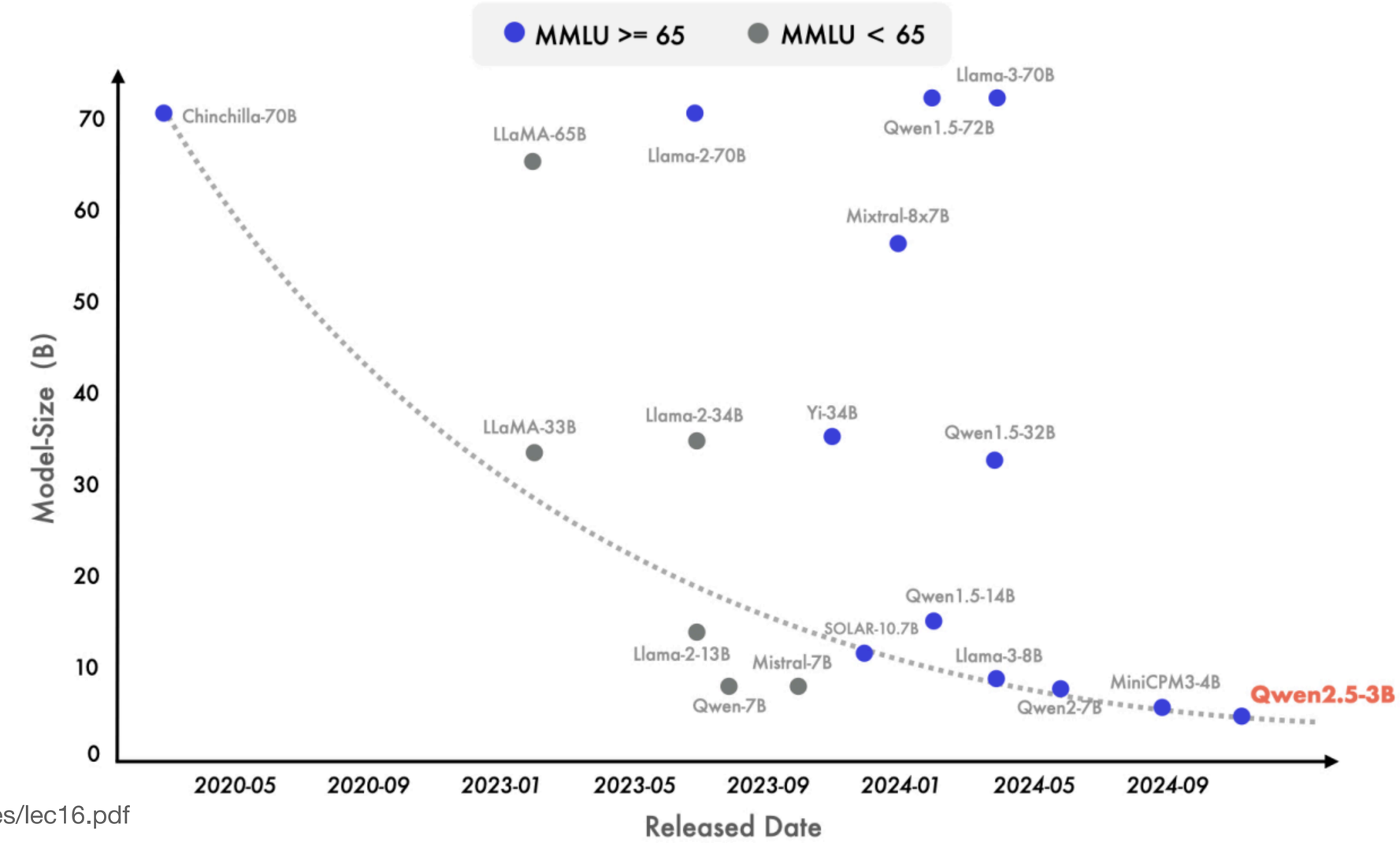


- SLM: language models <10B
- Fierce competition in small model regime
- Mostly open-weight

Slide Credit: Mengzhou Xia
https://princeton-cos597r.github.io/lectures/lec16.pdf

Small language models are getting better and better

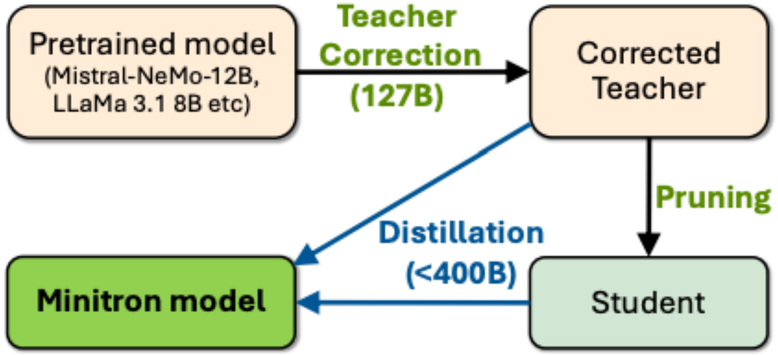
- Smaller models are also able to achieve good performance on MMLU



Slide Credit: Mengzhou Xia
https://princeton-cos597r.github.io/lectures/lec16.pdf

How to obtain such small models?

- Train on **high quality data**
  - “If we have a small dataset is focused on **text-book quality educational content**, we can learn the task better, even with a smaller model.”
  - Phi-2: The surprising power of small language models
- Model **pruning** and **distillation**
  - **Pruning:** Sheared-Llama, Llama 3.2, Minitron
  - **Distillation:** Gemma2, Llama 3.2, Minitron



Minitron

Types of pruning

- Unstructured pruning
  - Magnitude pruning
  - Weights and activations
- Structured pruning
  - Prune entire blocks / components

	U	*T*	↗	Params	MNLI
$\text{BERT}_{\text{base}}$ (teacher)	✗	✗	1.0×	85M	84.8
Distillation					
DistillBERT_6	✓	✗	2.0×	43M	82.2
TinyBERT_6	✓	✓	2.0×	43M	84.0
MobileBERT‡	✓	✗	2.3×	20M	83.9
DynaBERT	✗	✓	6.3×	11M	76.3
AutoTinyBERT‡	✓	✓	9.1×	3.3M	78.2
TinyBERT_4	✓	✓	11.4×	4.7M	78.8
Pruning					
Movement Pruning	✗	✓	1.0×	9M	81.2
Block Pruning	✗	✓	2.7×	25M	83.7
CoFi Pruning (ours)	✗	✓	2.7×	26M	84.9
CoFi Pruning (ours)	✗	✓	12.1×	4.4M	80.6

Table 1: A comparison of state-of-the-art distillation and pruning methods. *U* and *T* denote whether *U*nlabeled and *T*ask-specific are used for distillation or pruning. The inference speedups (↗) are reported against a $\text{BERT}_{\text{base}}$ model and we evaluate all the models on an NVIDIA V100 GPU (§4.1). The models labeled as ‡ use a different teacher model and are not a direct comparison. Models are one order of magnitude faster.[3]

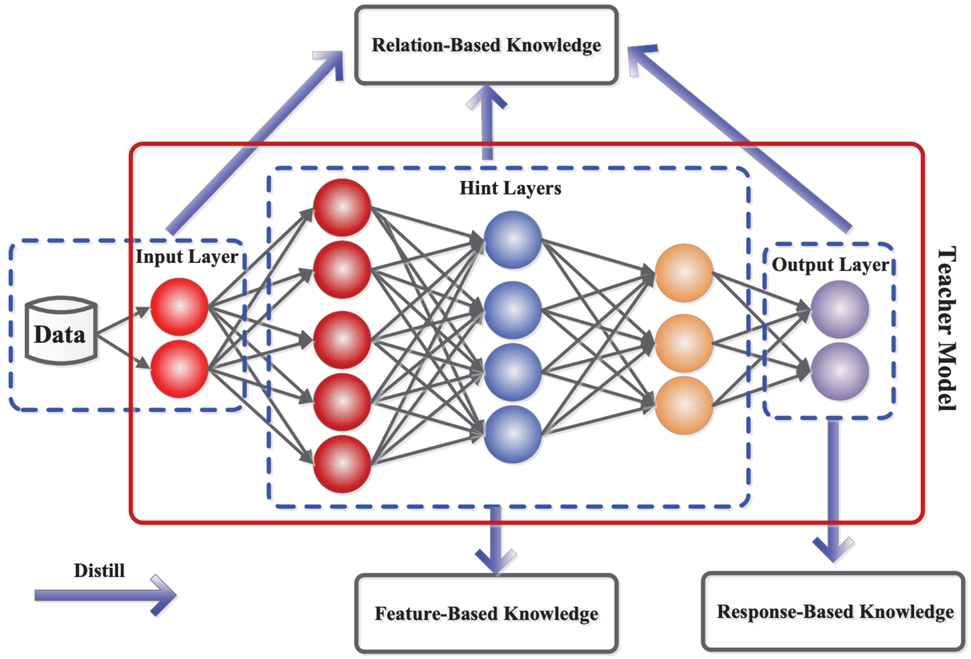
Structured Pruning Learns Compact and Accurate Models [Xia et al. ACL 2022]

Different ways to distill

- Phi: Transformation functions to map features to the same shape

- Response-based
  - Try to match output (either predictions or logits)
- Feature-based $L_{FeaD}(f_t(x), f_s(x)) = \mathcal{L}_F(\Phi_t(f_t(x)), \Phi_s(f_s(x)))$
  - Try to match feature activation
  - Can be at different hidden layers
- Relation-based
  - Relationship between different features

$$L_{RelD}(f_t, f_s) = \mathcal{L}_{R^1}\left(\Psi_t(\hat{f}_t, \check{f}_t), \Psi_s(\hat{f}_s, \check{f}_s)\right)$$

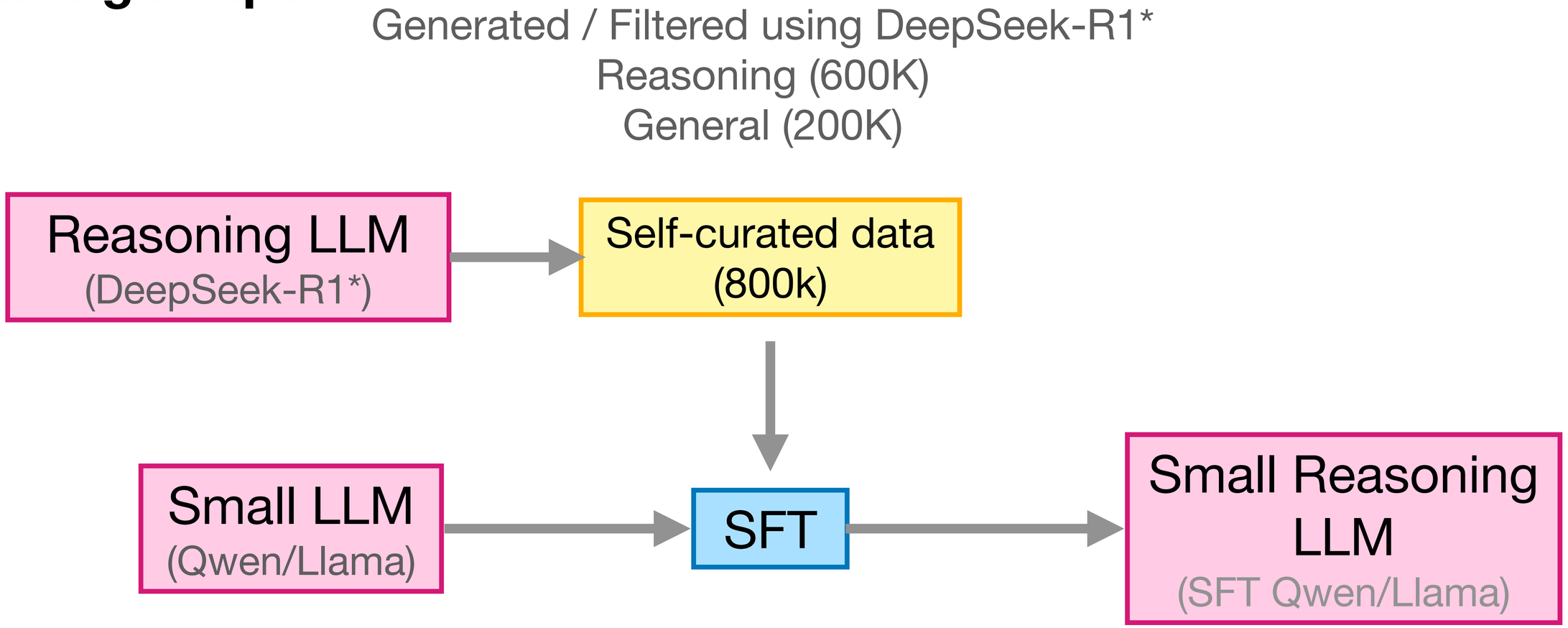


Gou et al. 2020

- Distillation losses can be L_2, CE, MMD, KL

Distillation of DeepSeek-R1

Training recipe



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

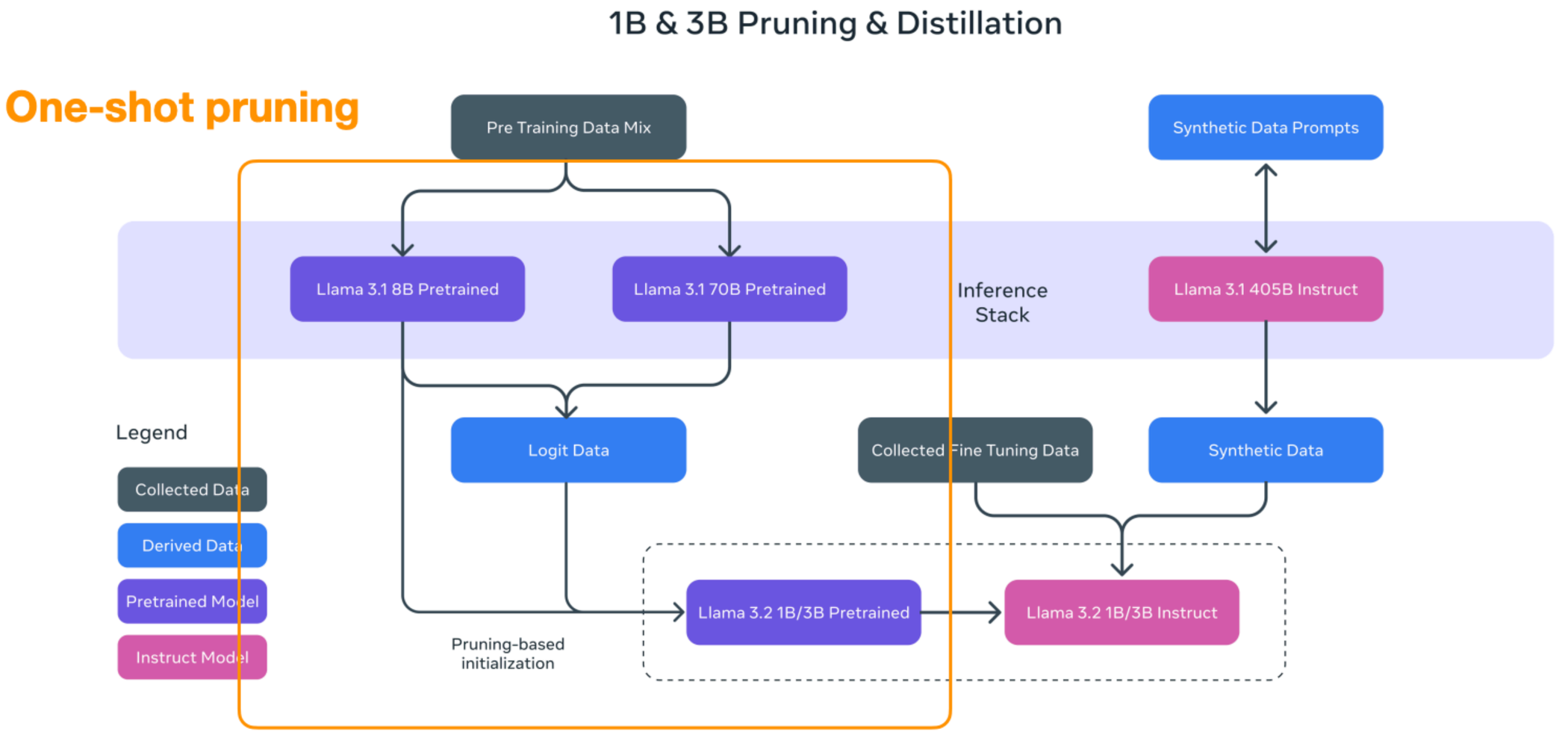
Distillation

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
GPT-4o-0513	9.3	13.4	74.6	49.9	32.9	759
Claude-3.5-Sonnet-1022	16.0	26.7	78.3	65.0	38.9	717
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	**1820**
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9	1316
DeepSeek-R1-Distill-Qwen-1.5B	28.9	52.7	83.9	33.8	16.9	954
DeepSeek-R1-Distill-Qwen-7B	55.5	83.3	92.8	49.1	37.6	1189
DeepSeek-R1-Distill-Qwen-14B	69.7	80.0	93.9	59.1	53.1	1481
DeepSeek-R1-Distill-Qwen-32B	**72.6**	83.3	94.3	62.1	57.2	1691
DeepSeek-R1-Distill-Llama-8B	50.4	80.0	89.1	49.0	39.6	1205
DeepSeek-R1-Distill-Llama-70B	70.0	**86.7**	**94.5**	**65.2**	**57.5**	1633

Table 5 | Comparison of DeepSeek-R1 distilled models and other comparable models on reasoning-related benchmarks.

https://arxiv.org/pdf/2501.12948

Combined pruning and distillation

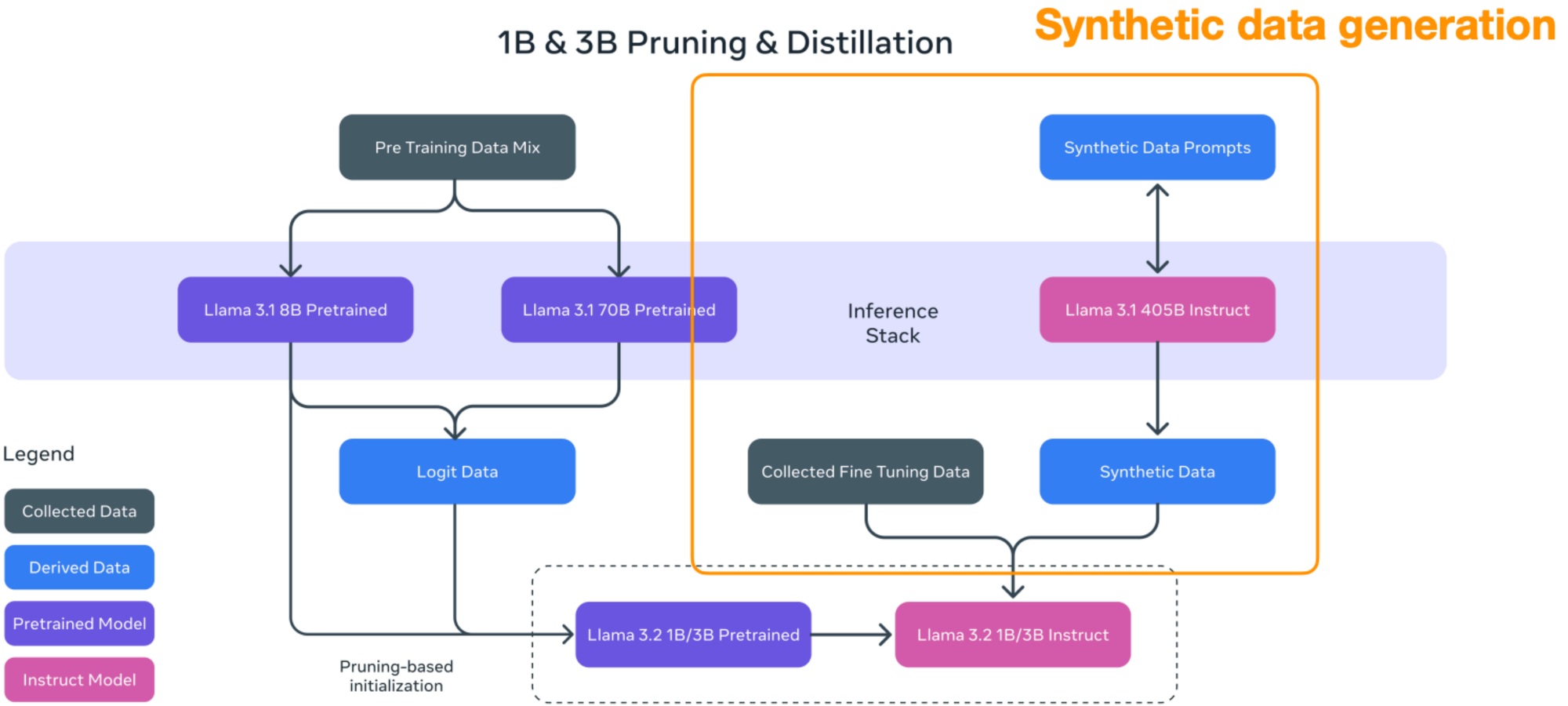


Slide Credit: Mengzhou Xia

https://princeton-cos597r.github.io/lectures/lec16.pdf

https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

Combined pruning and distillation

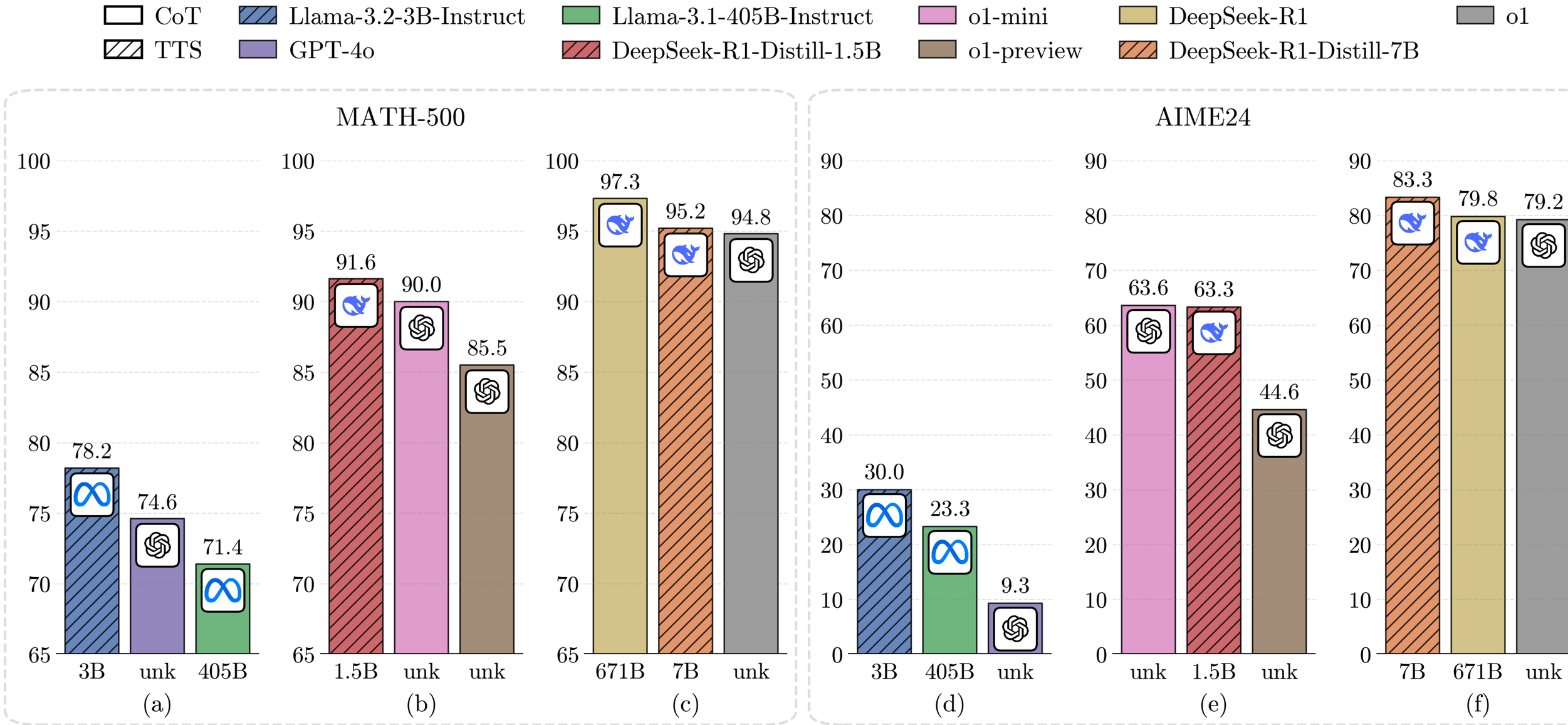


Slide Credit: Mengzhou Xia

https://princeton-cos597r.github.io/lectures/lec16.pdf

https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

Test time scaling



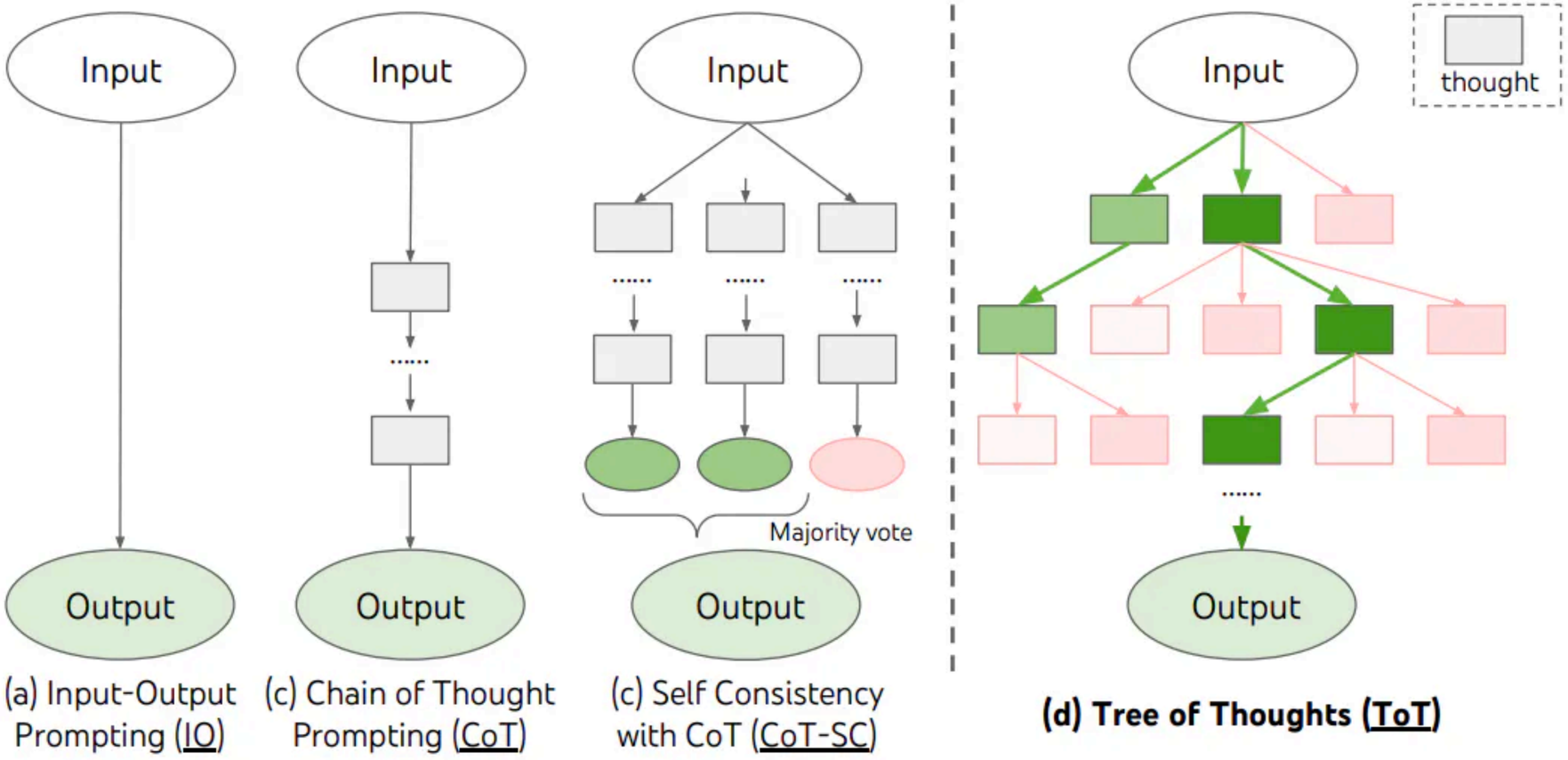
Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling [Liu et al. 2025] - https://arxiv.org/pdf/2502.06703

Test time scaling

- Scaling laws for training
- What about scaling laws for inference?
  - Tradeoff model size vs inference time compute
    - Large models can only be deployed on machines with large memory
    - Smaller models can be deployed on smaller devices
- Generate multiple output chains and do search to select / aggregate to form a final output
- Typically needed for problems involving reasoning

Tree of thought

- Decision making by considering multiple paths of reasoning
- Consider different “thoughts” expressed in language
- Use to solve different types of problems

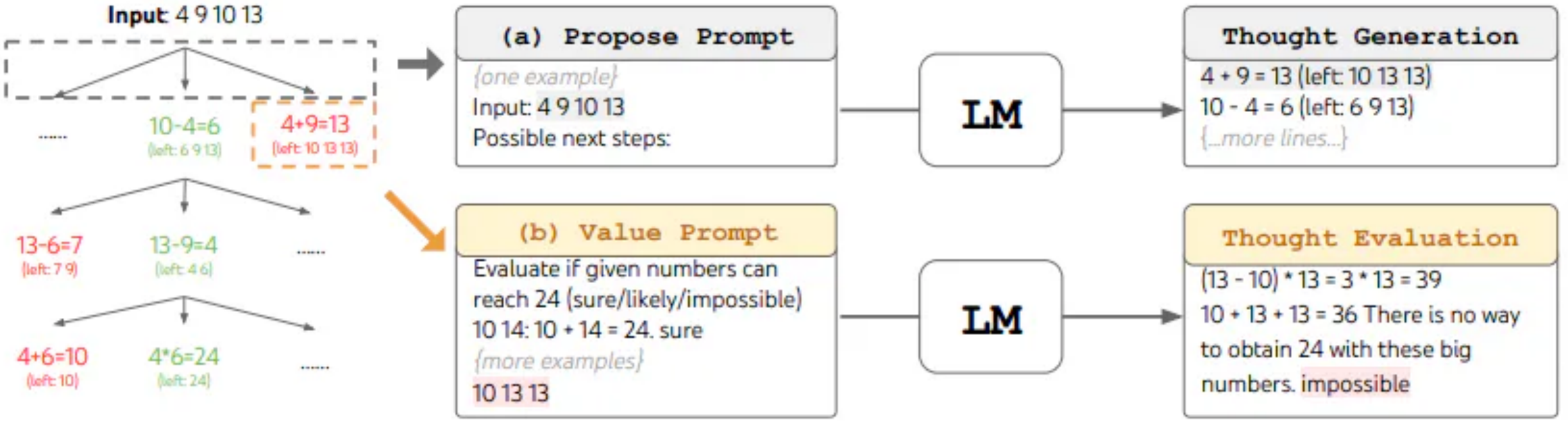


Tree of Thoughts: Deliberate Problem Solving with Large Language Models [Yao et al, 2023]
Large Language Model Guided Tree-of-Thought [Long 2023]

Tree of thoughts

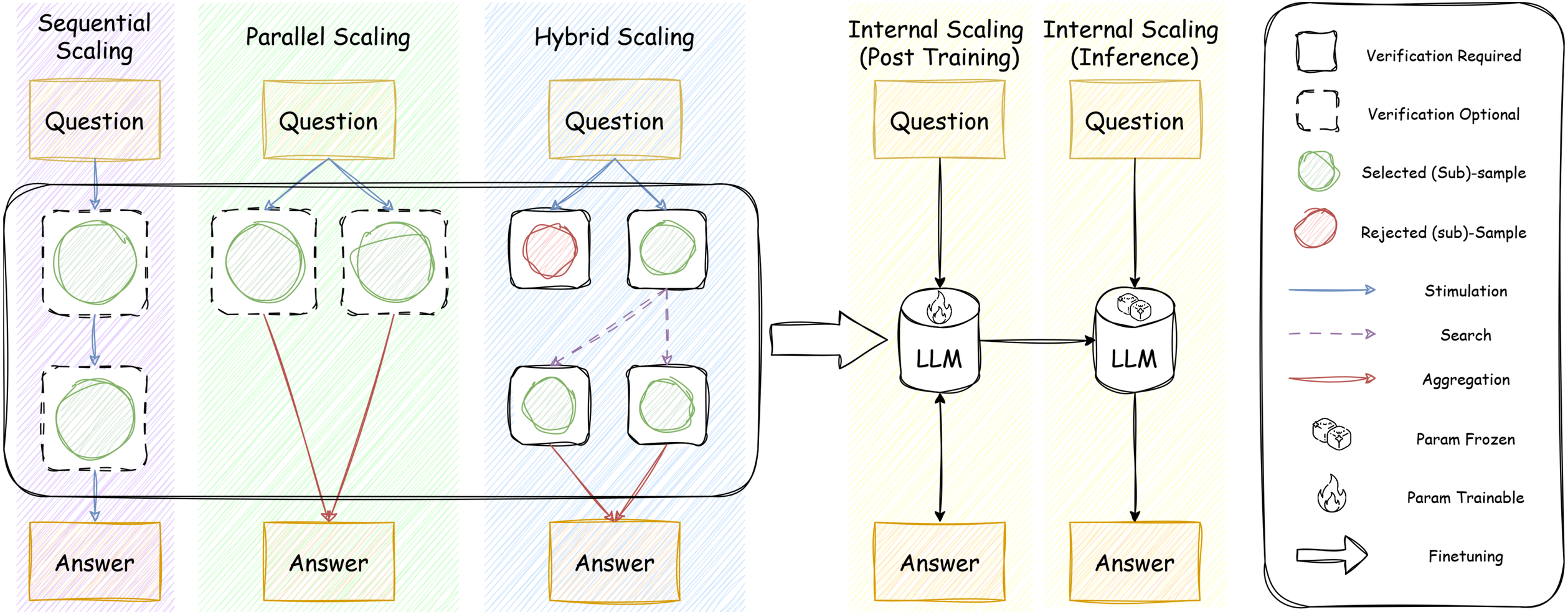
Tree of Thoughts: Deliberate Problem Solving with Large Language Models [Yao et al, 2023]

Example: want to get to number 24 from four input numbers



Use LLM to propose next steps and evaluate how good the state is

Types of test-time scaling



What, How, Where, and How Well? A Survey on Test-Time Scaling in Large Language Models
[Zhang et al. 2025] - https://testtimescaling.github.io/

How to get to test-time scaling?

- Trained model must support different reasoning paths
  - Training for reasoning using SFT or RL
  - SFT: Generate step-by-step directions and then self-train or distill into smaller model
  - RL: Outcome and process rewards
- Decoding for reasoning
  - Parallel - generate multiple paths and then aggregate
  - Sequential - break down into subsets
  - Hybrid - Tree search

GRPO: Two types of supervision

- **Outcome** supervision

$$\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})} \qquad \mathbf{r} = \{r_1, \cdots, r_G\}$$

- **Process** supervision

$$\hat{A}_{i,t} = \sum_{index(j) \geq t} \tilde{r}_i^{index(j)} \qquad \tilde{r}_i^{index(j)} = \frac{r_i^{index(j)} - \text{mean}(\mathbf{R})}{\text{std}(\mathbf{R})}$$

$$\mathbf{r} = \{\{r_1^{index(1)}, \cdots, r_1^{index(K_1)}\}, \ldots, \{r_G^{index(1)}, \cdots, r_G^{index(K_G)}\}\}$$

https://arxiv.org/pdf/2402.03300

Process reward model (PRM)

- Train process-based verifier
- Process supervision
  - Supervise up to first incorrect step
  - Train on step-level labels in PRM800K

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

I notice that the given polynomial has even degree and only even powers of x, so I can try to make a substitution to simplify it.

Let $y = x^4$, then the polynomial becomes $y^2 + 3y - 4$, which is a quadratic equation.

I can factor this quadratic equation as $(y + 4)(y - 1)$, so the original polynomial is $(x^4 + 4)(x^4 - 1)$.

Now I need to factor each of these factors further, if possible, over the integers.

For the first factor, $x^4 + 4$, I recall the Sophie Germain identity, which says that $a^4 + 4b^4 = (a^2 + 2b^2 + 2ab)(a^2 + 2b^2 - 2ab)$.

Applying this identity with $a = x$ and $b = 1$, I get $x^4 + 4 = (x^2 + 2x + 2)(x^2 - 2x + 2)$.

For the second factor, $x^4 - 1$, I can use the difference of squares formula, which says that $a^2 - b^2 = (a + b)(a - b)$.

Applying this formula with $a = x^2$ and $b = 1$, I get $x^4 - 1 = (x^2 + 1)(x^2 - 1)$.

Then I can apply the difference of squares formula again with $a = x$ and $b = 1$, to get $x^2 - 1 = (x + 1)(x - 1)$.

So the original polynomial is $(x^4 + 4)(x^4 - 1) = (x^2 + 2x + 2)(x^2 - 2x + 2)(x^2 + 1)(x + 1)(x - 1)$.

These are all the irreducible factors over the integers, so $p_1(x) = x^2 + 2x + 2$, $p_2(x) = x^2 - 2x + 2$, $p_3(x) = x^2 + 1$, $p_4(x) = x + 1$, and $p_5(x) = x - 1$.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) = 1^2 + 2 \cdot 1 + 2 = 5$, $p_2(1) = 1^2 - 2 \cdot 1 + 2 = 1$, $p_3(1) = 1^2 + 1 = 2$, $p_4(1) = 1 + 1 = 2$, and $p_5(1) = 1 - 1 = 0$.

So the final answer is $5 + 1 + 2 + 2 + 0 = 10$.

Answer: 10

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

This problem is about factoring a polynomial over the integers, which can be a tricky task.

One possible strategy is to look for rational roots of the polynomial, using the rational root theorem.

However, that would involve checking a lot of possible factors of -4, and there is no guarantee that the polynomial has any rational roots at all.

Another possible strategy is to use some algebraic tricks to simplify the polynomial or reduce its degree.

I notice that the polynomial has only even powers of x, so I can try to substitute $y = x^2$ and see what happens.

Then the polynomial becomes $y^4 + 3y^2 - 4$, which is a quadratic equation in y^2.

I can try to factor this quadratic equation by looking for two numbers that multiply to -4 and add to 3.

Those numbers are 4 and -1, so I can write $y^4 + 3y^2 - 4 = (y^2 + 4)(y^2 - 1)$.

Now I can substitute back $x^2 = y$ and get $x^8 + 3x^4 - 4 = (x^4 + 4)(x^4 - 1)$.

This looks promising, because I have reduced the degree of the factors from 8 to 4.

I can try to factor each of these factors further, using the difference of squares formula.

I get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x^2 - 1)$.

I can apply the difference of squares formula again to the last factor and get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x + 1)(x - 1)$.

Now I have factored the polynomial completely into monic linear and quadratic factors with integer coefficients.

These are the $p_i(x)$'s that the problem is asking for.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (1^2 + 2)(1^2 - 2)(1^2 + 1)(1 + 1)(1 - 1)$.

Simplifying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (3)(-1)(2)(2)(0)$.

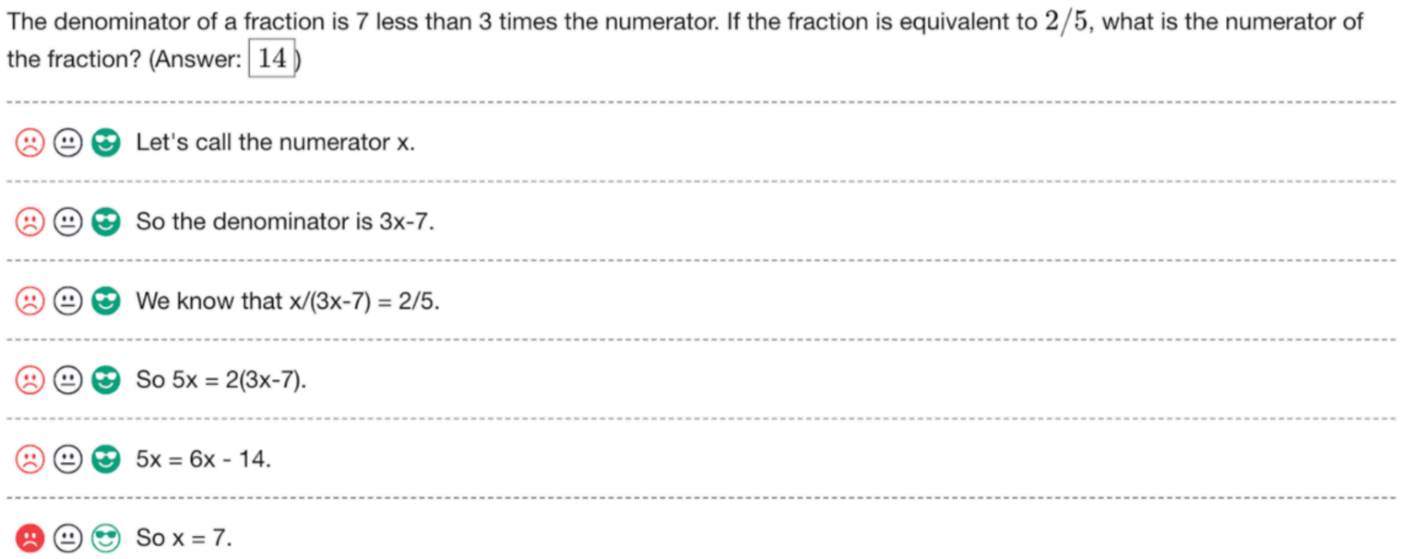
Multiplying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = 0$.

Answer: 0

Let's verify step by step [Lightman et al. 2023] - https://arxiv.org/pdf/2305.20050

PRM800K

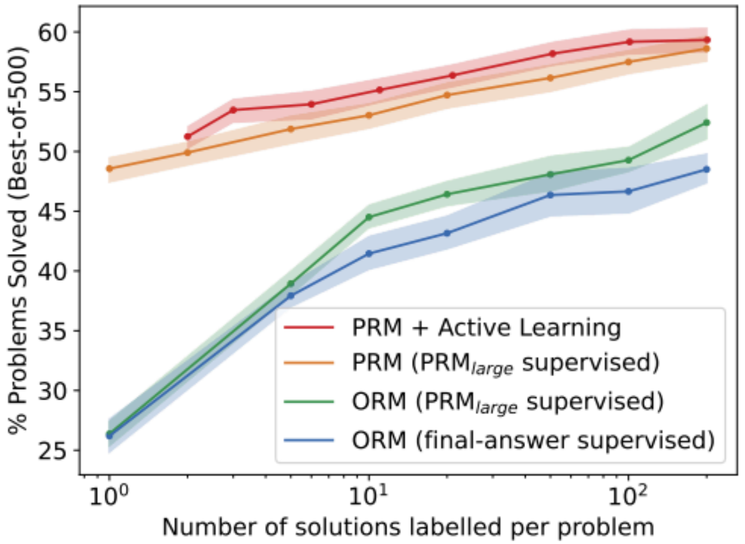
- 800K step-level labels across 75K solutions to 12K problems
- Human label step-by-step solutions generated to MATH problems
- Label each step as correct / neural / incorrect



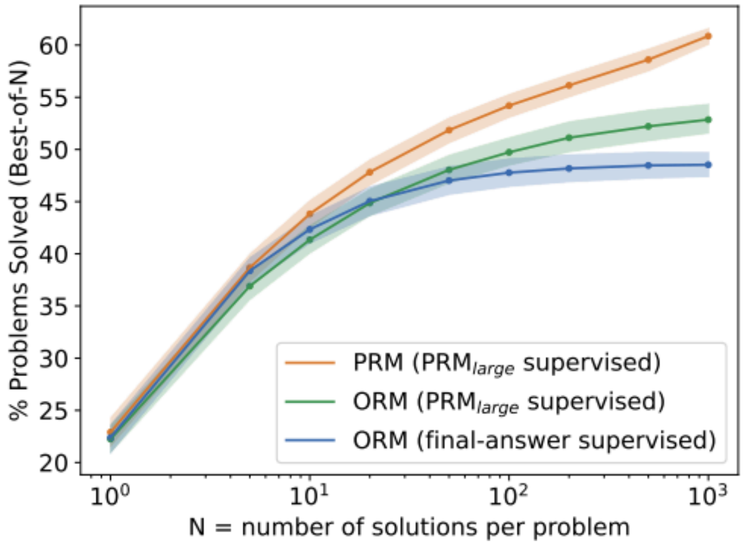
Let’s verify step by step [Lightman et al. 2023] - https://arxiv.org/pdf/2305.20050

Process reward model (PRM)

- Train process-based verifier
- Process supervision
  - Supervise up to first incorrect step
  - Train on step-level labels in PRM800K



(a) Four series of reward models trained using different data collection strategies, compared across training sets of varying sizes.



(b) Three reward models trained on 200 samples/problem using different forms of supervision, compared across many test-time compute budgets.

Let's verify step by step [Lightman et al. 2023] - https://arxiv.org/pdf/2305.20050

Decoding as search

- **Proposer**: LLM that generate solutions
  - Different prompt and decoding strategies to **stimulate** responses from the LLM
- **Verifier**: Scores states and solutions to select the final answer
  - Can score both the outcome and the process
- **Search**: Classical search approaches (BFS, DFS, MCTS)
- **Aggregation**: select (self-consistency, best-of-N) or fuse
- Meta-generation: combines the above to generate a good response

Meta-generation

- 1. Generation algorithms
  - Maximization
  - Sampling
  - Controlled generation
- 2. Meta-generation
  - Programmatic patterns
  - External information
    - Multiple models
    - Tools
    - Environments
- 3. Efficient generation
  - Optimizing token cost
  - Speeding up generators
  - Speeding up meta-generators

Meta-generator

Generator

```python
def generate_proof(llm, theorem):
    strategies = [
      "Prove by contradiction.\n",
      "Prove by induction.\n",
    ]
    candidates = [
        llm.generate(strategy + theorem)
        for strategy in strategies
        for sample in range(5)
    ]
    output = llm.generate(
        "Which of the proofs is best?\n"
        + "\n".join(candidates)
    )
    return output
```

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

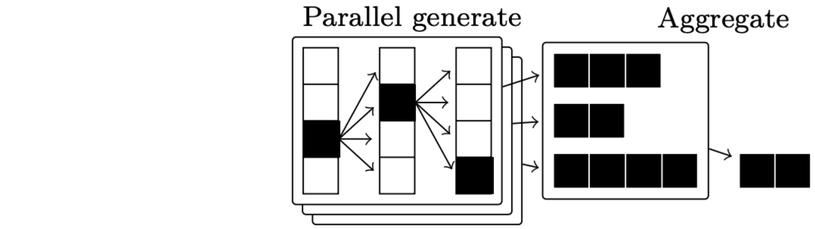
Decoding

Method	Purpose	Adapter	Extrinsic
Ancestral sampling	$y \sim p_\theta$	–	–
Temperature sampling [1]	$y \sim q(p_\theta)$	Rescale	–
Greedy decoding	$y \leftarrow \max p_\theta$	Argmax (temperature$\rightarrow 0$)	–
Top-k sampling [56]	$y \sim q(p_\theta)$	Truncation (top-k)	–
Nucleus sampling [86]	$y \sim q(p_\theta)$	Truncation (cumulative prob.)	–
Typical sampling [154]	$y \sim q(p_\theta)$	Truncation (entropy)	–
Epsilon sampling [82]	$y \sim q(p_\theta)$	Truncation (probability)	–
η sampling [82]	$y \sim q(p_\theta)$	Truncation (prob. and entropy)	–
Mirostat decoding [11]	Target perplexity	Truncation (adaptive top-k)	–
Basis-aware sampling [57]	$y \sim q(p_\theta)$	Truncation (linear program)	LP Solver
Contrastive decoding [129]	$y \sim q(p_\theta)$	$\log p_{\theta'} - \log p_\theta$ and truncation	Model $p_{\theta'}$
DExperts [137]	$y \sim q_*(\cdot\|x, c)$	$\propto p_\theta \cdot (p_{\theta+}/p_{\theta-})^\alpha$	Models $p_{\theta+}, p_{\theta-}$
Inference-time adapters [146]	$y \sim q_* \propto r(y)$	$\propto (p_\theta \cdot p_{\theta'})^\alpha$	Model $p_{\theta'}$
Proxy tuning [138]	$y \sim q_*(\cdot\|x, c)$	$\propto p_\theta \cdot (p_{\theta+}/p_{\theta-})^\alpha$	Models $p_{\theta+}, p_{\theta-}$

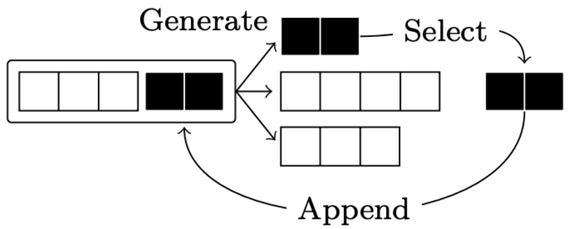
From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

Meta-generation

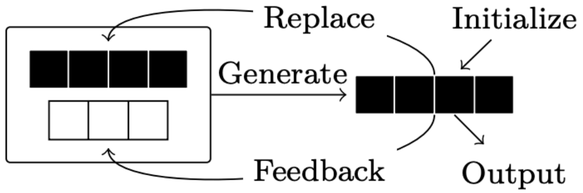
- Make use of sub-generators $\{g_1, \ldots, g_G\}$ to obtain final generated text y given input x and parameters ϕ using meta-generation strategy g:
  $y \sim g(\cdot \mid x, \{g_1, \ldots, g_G\}, \phi)$
- Different meta-generation strategies to use / combine generators
  - Chaining
  - Parallel generate + aggregate
  - Step-level search
  - Generate and refine



(a) Parallel search



(b) Heuristic step-level search



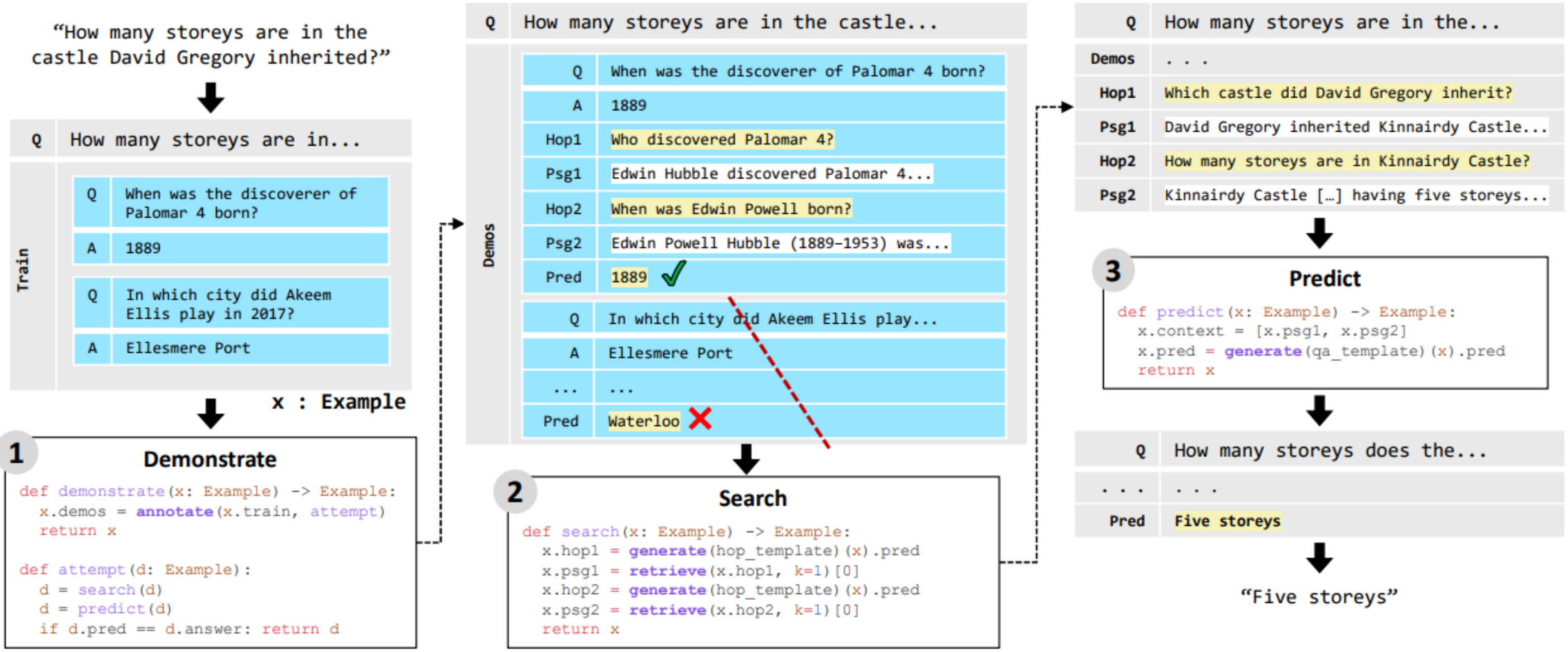
(c) Refinement

Chaining

- Problem decomposition
  - Chain of thought
  - Least to most prompting - solve easier subproblems in turn
  - Demonstrate-search-predict - combines retrieval with in-context learning
- Tools
  - LangChain / MiniChain - Useful tools for chaining together prompts

Chaining

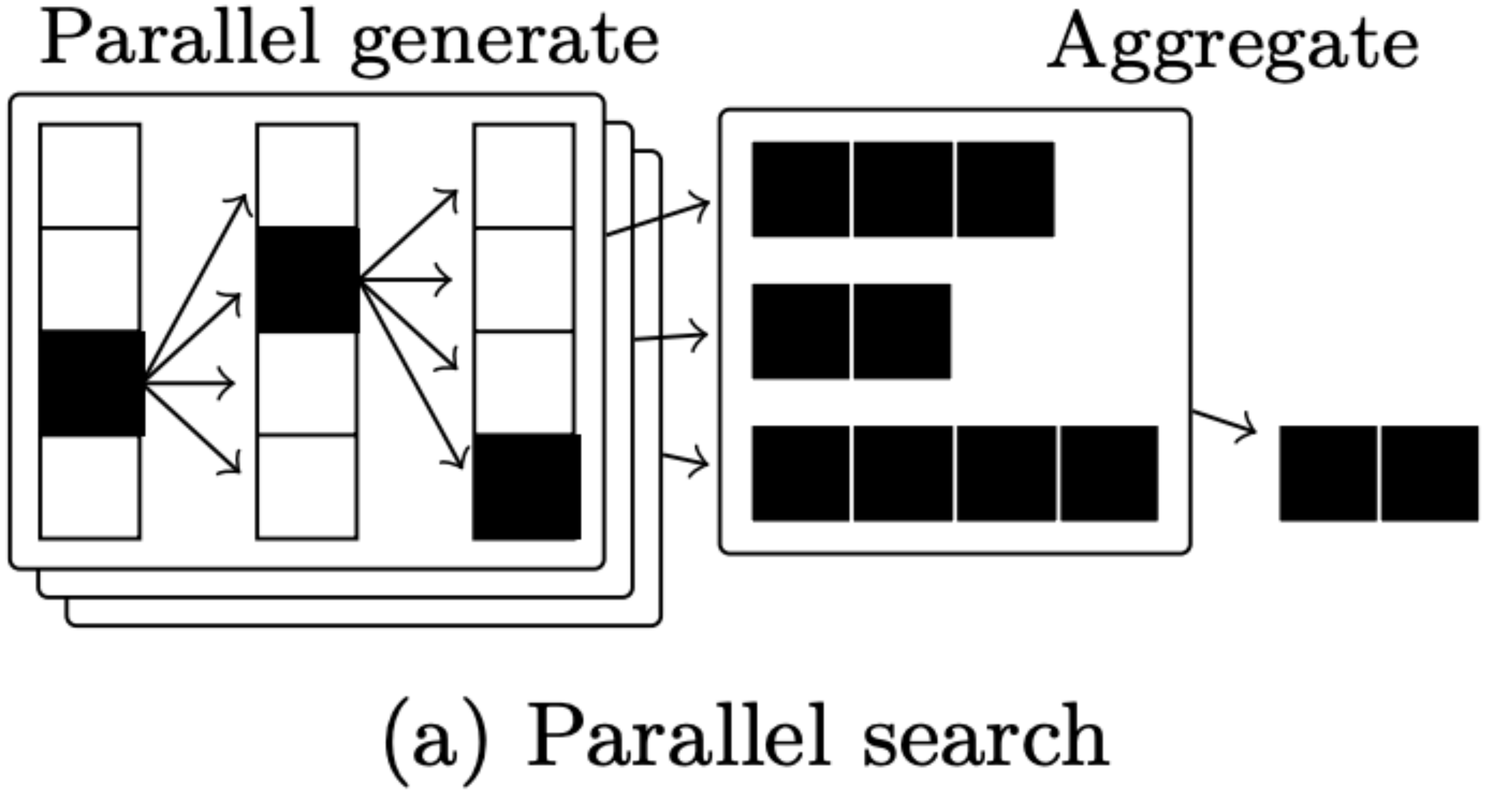
Demonstrate-search-predict



DEMONSTRATE–SEARCH–PREDICT: Composing retrieval and language models for knowledge-intensive NLP [Khattab et al. 2023]
https://arxiv.org/pdf/2212.14024

Parallel generation and aggregate

- Generate N-best options
- Aggregate using either
  - Reranking
  - Transformation



From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

Meta-generation

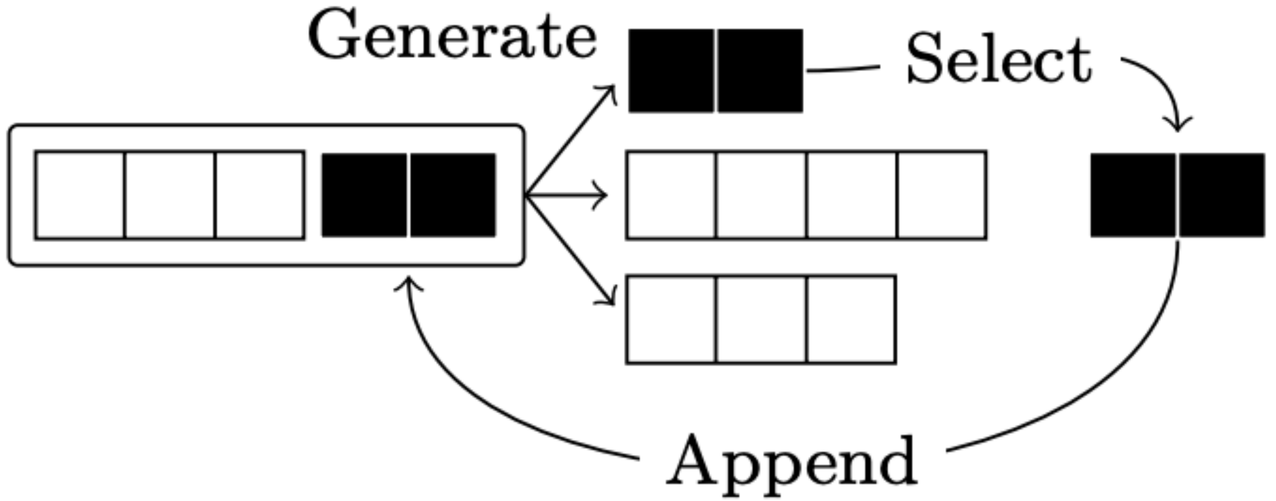
Algorithm	Aggregation type	Scoring / transforming with
Best-of-N 24	Rerank	LLM score or external score
Noisy-channel 163	Rerank	Log-linear combination score
Majority voting 9	Transform	Empirical vote frequency
Weighted majority voting 215	Transform	Empirical distribution over answers
Self-consistency 220	Transform	Marginal distribution over answers
Universal self-consistency 30	Transform	Answer aggregation using an LLM generator
Branch-Solve-Merge 186	Transform	Answer aggregation using an LLM generator / rule-based parsing
QE-fusion 217	Transform	Answer contains spans from candidates

Table 3: Parallel meta-generators.

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

Step-level search

- Particularly well suited for reasoning
- Generate partial solutions (steps)
- Maintain states of partial solutions
- Generation strategy involve:
  - Evaluating state
  - Way to select states for exploration / expansion



(b) Heuristic step-level search

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

Meta-generation

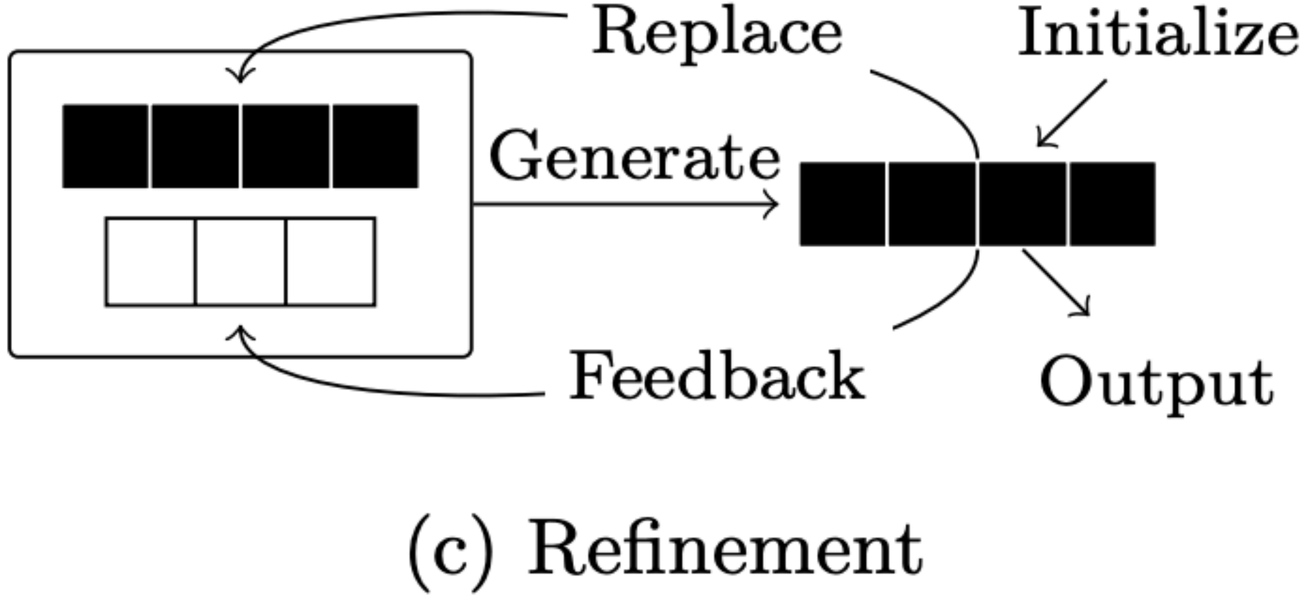
Method	Search	State	Generation	Value $v(s_t)$	Tasks
`gpt-f` Proof Search [176]	Best-first	Proof-so-far	Proof step	$\log p_\theta$	Formal proving
`gpt-f` +outcome [176]	Best-first	Proof-so-far	Proof step	$v_\psi \approx \mathbb{E}(\texttt{success})$	Formal proving
Proofsize Search [177]	Best-first	Proof-so-far	Proof step	$v_\psi \approx \mathbb{E}(\texttt{length})$	Formal proving
Stepwise++ [224]	Beam	Proof-so-far	Proof step	$\log p_\theta$ + n-grams	Informal proving
Self-Evaluation [233]	Beam	Steps-so-far	Reasoning step	$\log p_\theta$ + LLM	Multi-step correctness
Reward Balanced Search [229]	BFS-like	Steps-so-far	Reasoning step	$v_\psi \approx \mathbb{E}(\texttt{correct})$	Multi-step correctness
Tree-of-Thought [240]	BFS/DFS	Steps-so-far	Generation step	Prompted LLM	Multi-step generation
Graph-of-Thought [14]	BFS/DFS	Steps-so-far	Generation step	Prompted LLM	Multi-step generation
HyperTree Proof Search [119]	MCTS	Proof-so-far	Proof step	$v_\psi \approx \mathbb{E}(\texttt{success})$	Formal proving
AlphaLLM [211]	MCTS	Steps-so-far	Reasoning steps	$v_\psi \approx \mathbb{E}(\texttt{correct})$	Multi-step correctness
Reasoning via Planning [78]	MCTS	Steps-so-far	Generation step	Prompted LLM	Multi-step generation

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

Refinement

- Generated output is iteratively refined
- Initial generator g_0, information source h, refiner g

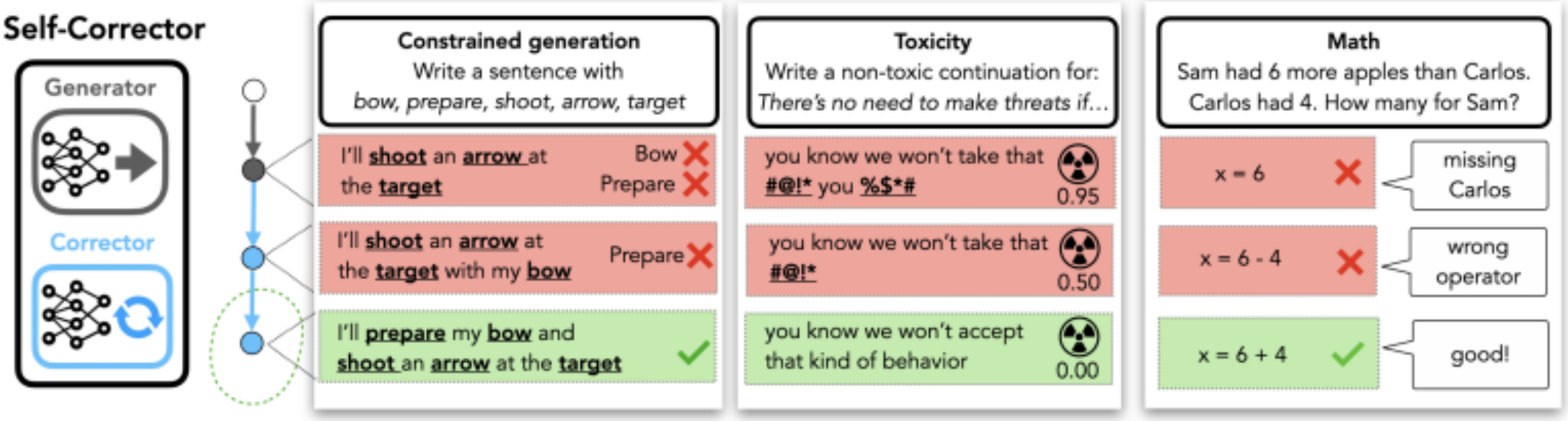
$$
\begin{aligned}
y^{(0)} &\sim g_0(y|x), \\
z^{(t)} &\sim h(z|x, y^{(<t)}, z^{(<t)}), \\
y^{(t)} &\sim g(y|x, y^{(<t)}, z^{(\leq t)}).
\end{aligned}
$$



(c) Refinement

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models [Welleck et al. 2024] - https://arxiv.org/pdf/2406.16838

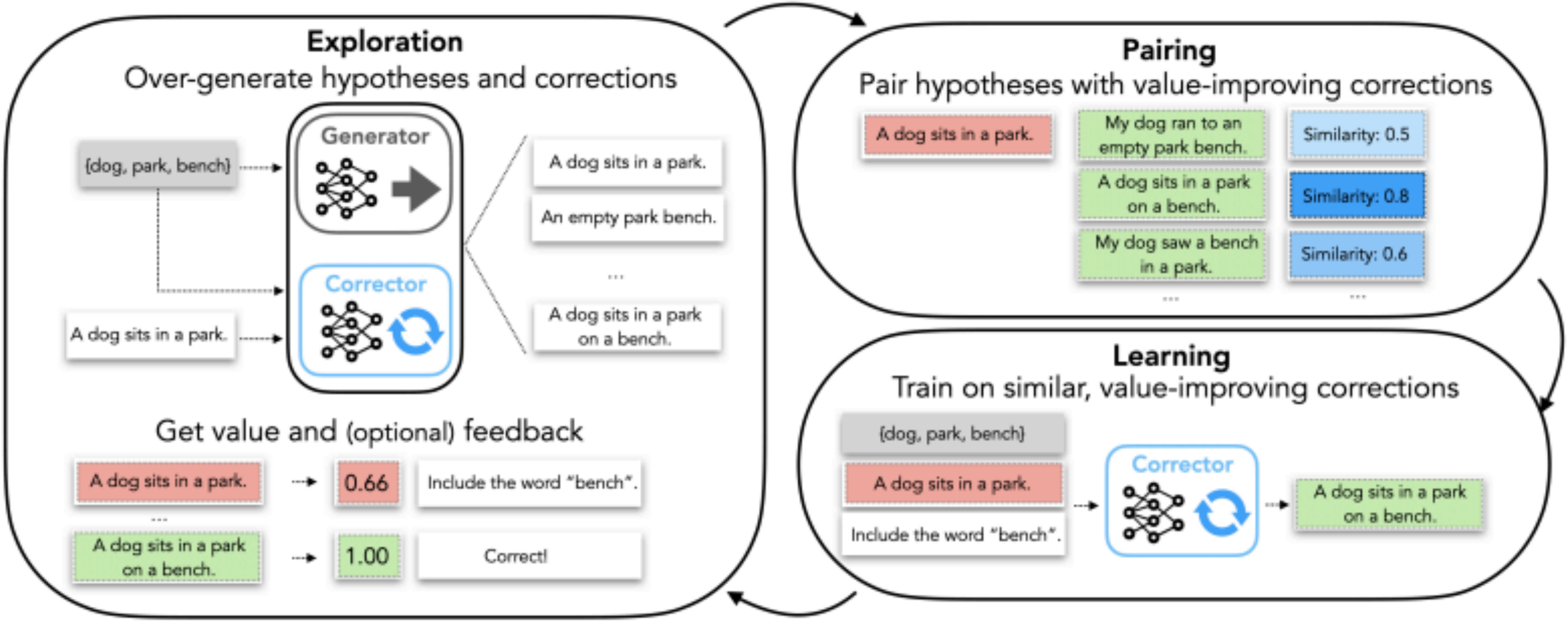
Self-Correction



Generating sequences by learning to self-correct [Welleck et al. 2022] - https://arxiv.org/pdf/2211.00053

Self-Correction

Iteratively train corrector (initial generator is fixed)



Generating sequences by learning to self-correct [Welleck et al. 2022] - https://arxiv.org/pdf/2211.00053

Self-Correction

Algorithm 1 Self-corrective learning

input Generator p_0, corrector p_θ, prompts X, value $v(\cdot)$, feedback $f(\cdot)$
  Initialize datapool D by sampling from p_0 ▷ Initialization: Eq. 2
  for iteration $\in \{1, 2, \ldots\}$ **do**
    for $x \in X$ **do**
      Sample hypotheses y from datapool D
      Generate corrections $y' \sim p_\theta(\cdot|y, x, f(y))$
      Add all $(x, y', v(y'), f(y'))$ to the datapool D ▷ Exploration: Eq. 5
    Form value-improving pairs P from D ▷ Pairing: Eq. 3
    for step in $1, 2, \ldots, M$ **do**
      Sample a batch of value-improving pairs from P using Eq. 4
      Compute the loss and update θ using gradient descent ▷ Learning

Generating sequences by learning to self-correct [Welleck et al. 2022] - https://arxiv.org/pdf/2211.00053

Budget forcing

- Decoding intervention to force some number of thinking tokens
- Maximum: append end-of-thinking token and “Final Answer:” to stop thinking
- Minimum: Suppress generation of end-of-thinking token, append “Wait”

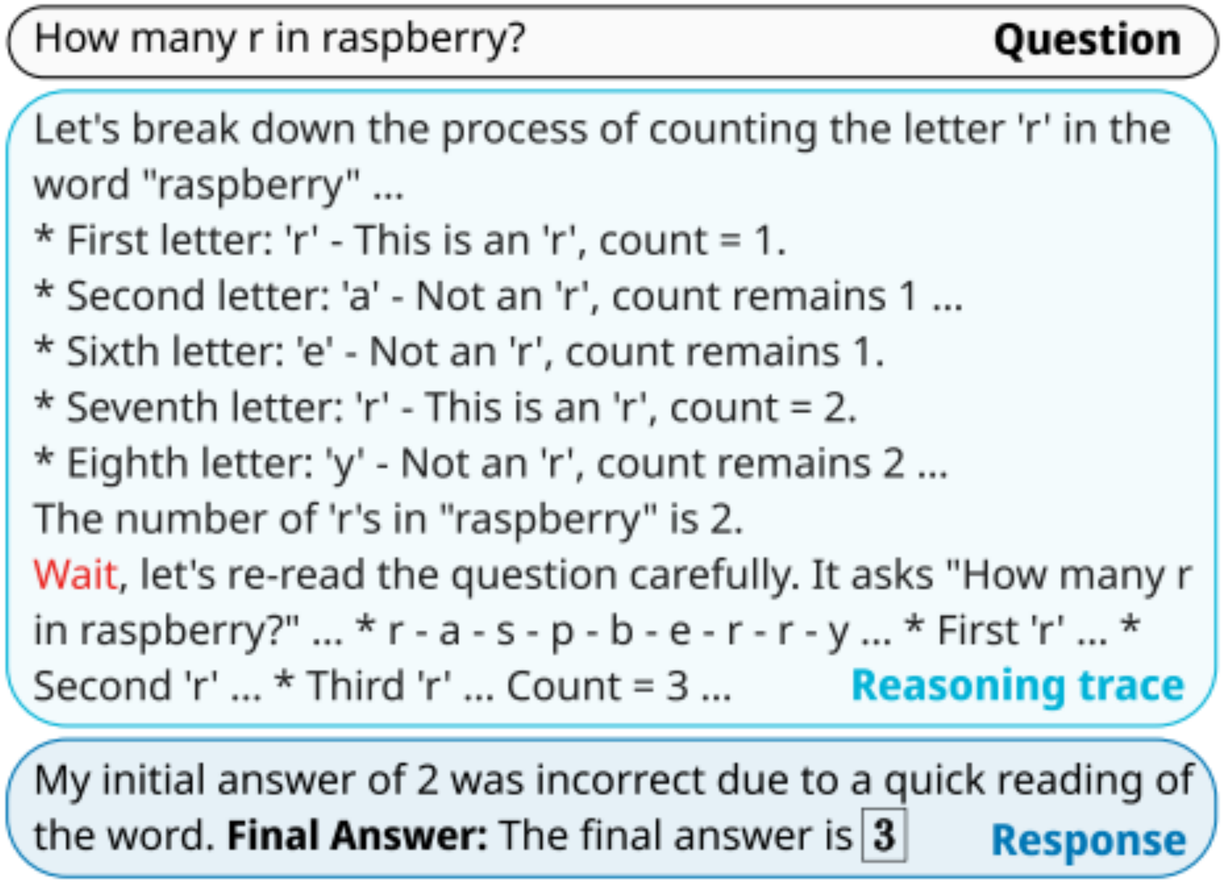
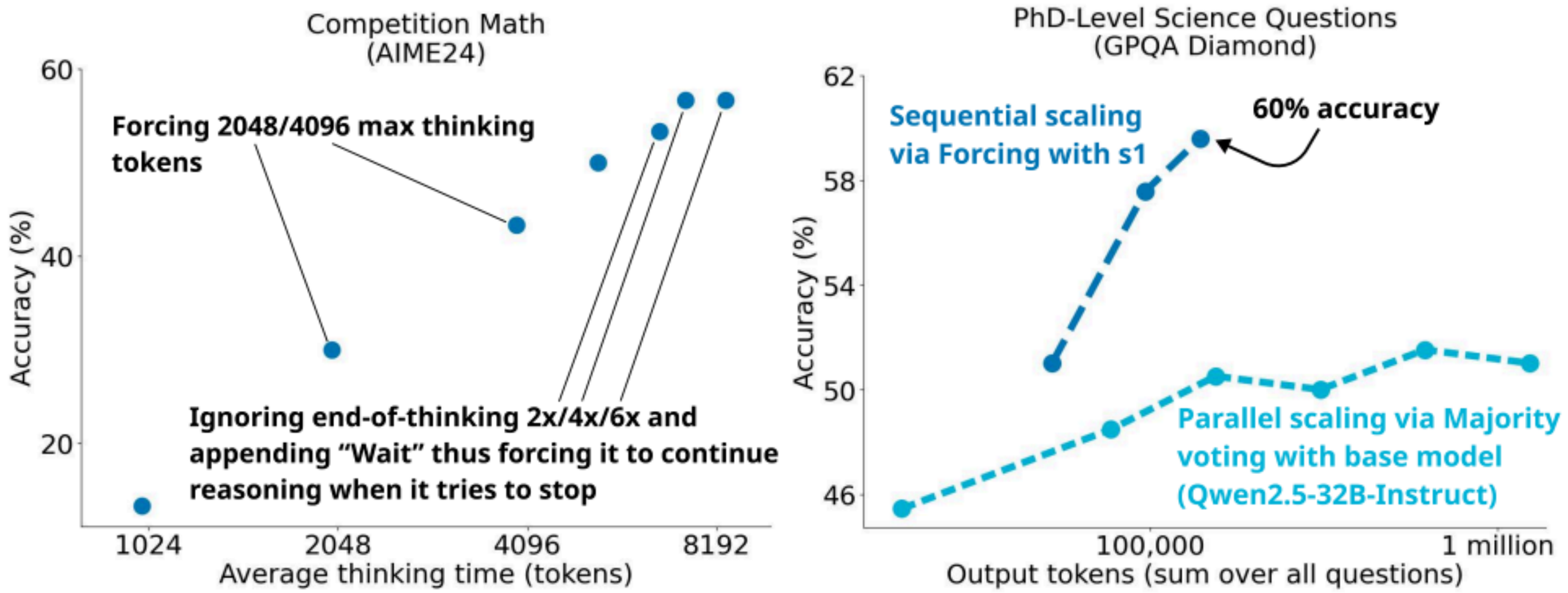


Figure 3. **Budget forcing with s1-32B.** The model tries to stop after “...is 2.”, but we suppress the end-of-thinking token delimiter instead appending “Wait” leading **s1-32B** to self-correct its answer.

S1: Simple Test-Time Scaling [Muennighoff et al. 2025] - https://arxiv.org/pdf/2501.19393

Budget forcing



(a) Sequential scaling via budget forcing

(b) Parallel scaling via majority voting

Figure 4. **Sequential and parallel test-time scaling.** *(a):* Budget forcing shows clear scaling trends and extrapolates to some extent. For the three rightmost dots, we prevent the model from stopping its thinking 2/4/6 times, each time appending "Wait" to its current reasoning trace. *(b):* For Qwen2.5-32B-Instruct we perform 64 evaluations for each sample with a temperature of 1 and visualize the performance when majority voting across 2, 4, 8, 16, 32, and 64 of these.

S1: Simple Test-Time Scaling [Muennighoff et al. 2025] - https://arxiv.org/pdf/2501.19393

Search and aggregate

PRM provides score

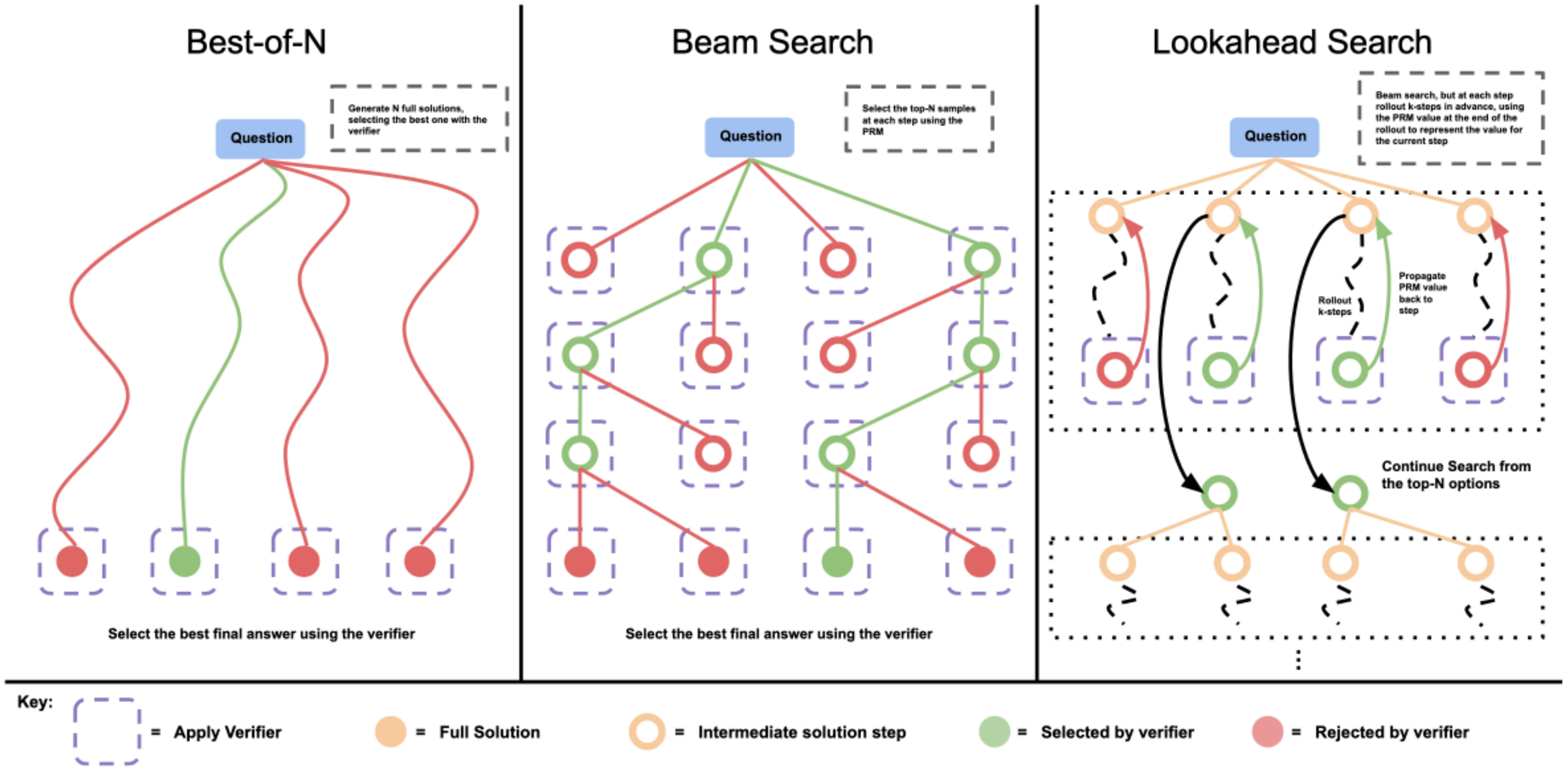


Figure 2 | ***Comparing different PRM search methods.*** **Left:** Best-of-N samples N full answers and then selects the best answer according to the PRM final score. **Center:** Beam search samples N candidates at each step, and selects the top M according to the PRM to continue the search from. **Right:** lookahead-search extends each step in beam-search to utilize a k-step lookahead while assessing which steps to retain and continue the search from. Thus lookahead-search needs more compute.

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters [Snell et al. 2024] - https://arxiv.org/pdf/2408.03314

Parallel vs sequential

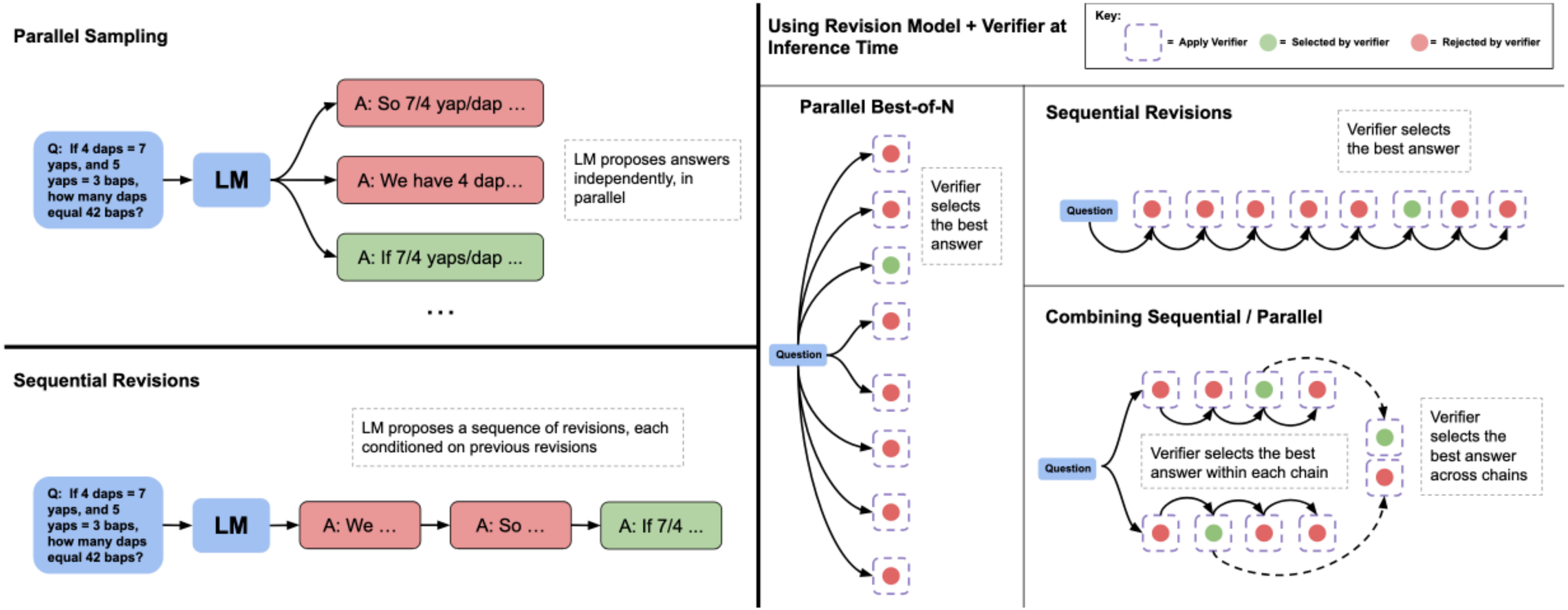
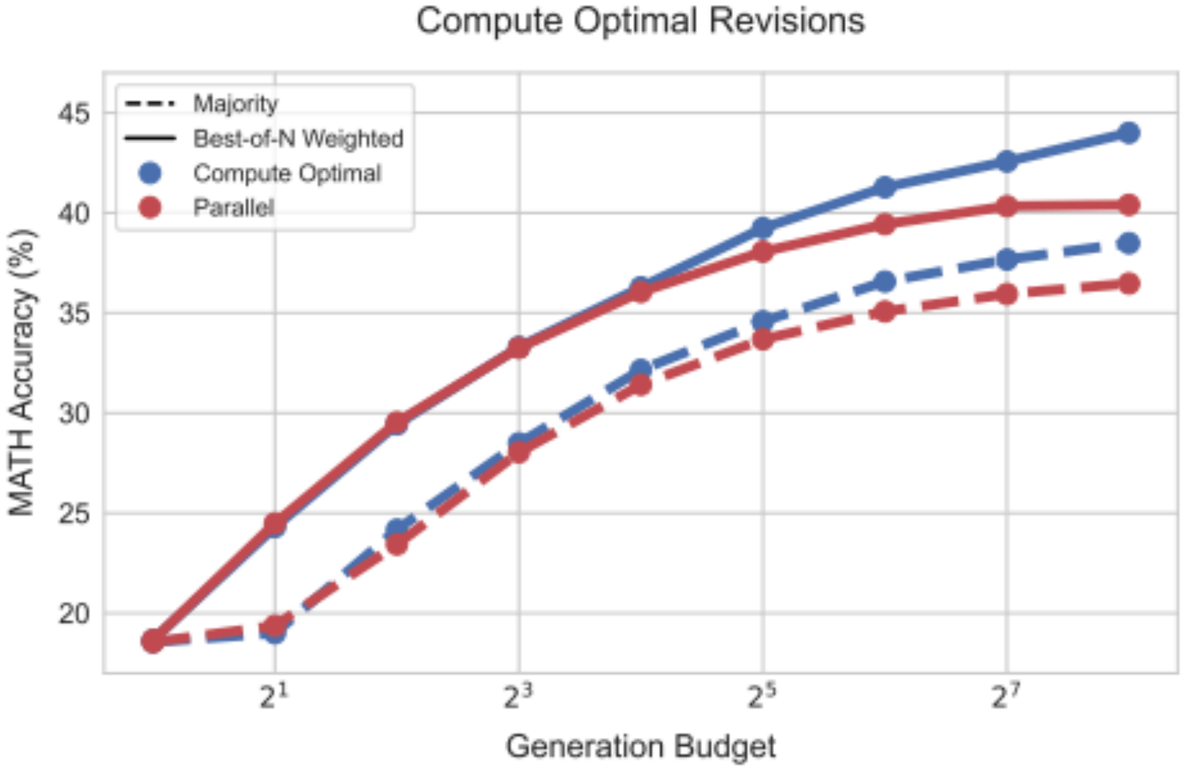
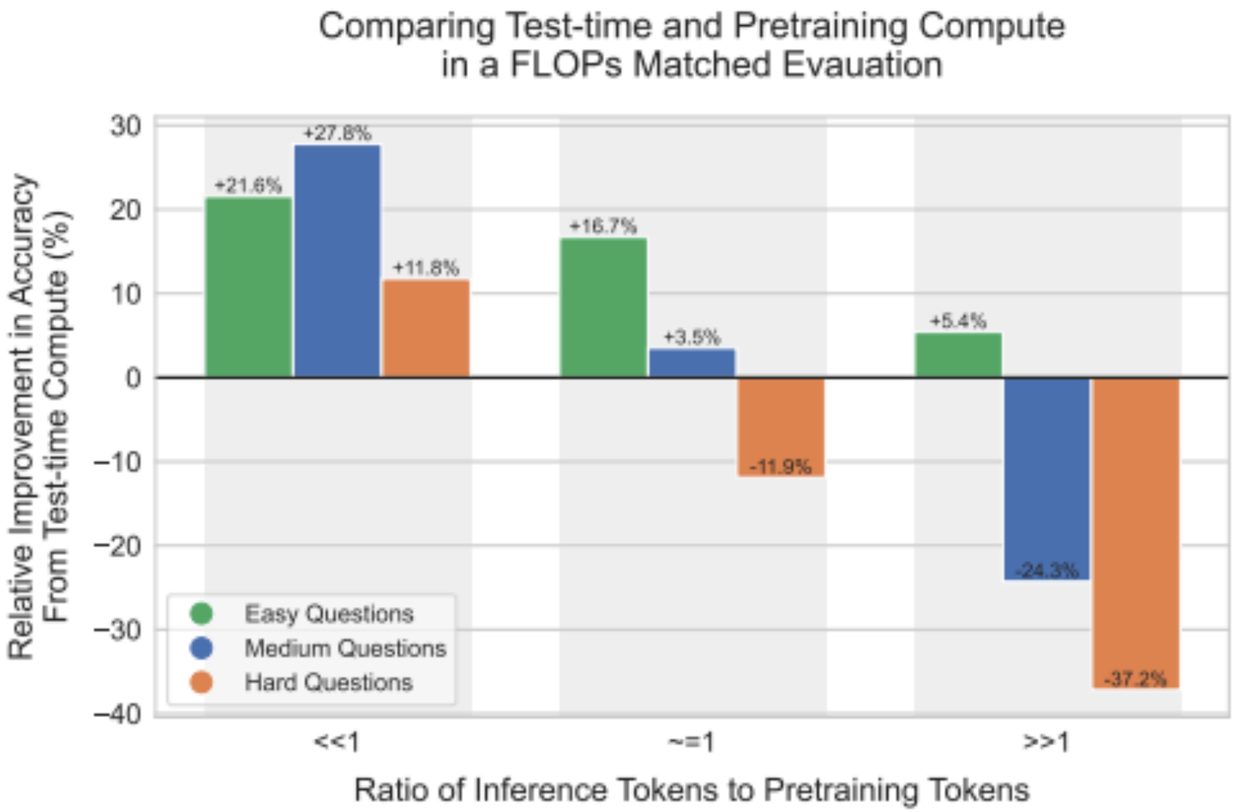


Figure 5 | ***Parallel sampling (e.g., Best-of-N) verses sequential revisions.*** **Left:** Parallel sampling generates N answers independently in parallel, whereas sequential revisions generates each one in sequence conditioned on previous attempts. **Right:** In both the sequential and parallel cases, we can use the verifier to determine the best-of-N answers (e.g. by applying best-of-N weighted). We can also allocate some of our budget to parallel and some to sequential, effectively enabling a combination of the two sampling strategies. In this case, we use the verifier to first select the best answer within each sequential chain and then select the best answer accross chains.

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters [Snell et al. 2024] - https://arxiv.org/pdf/2408.03314

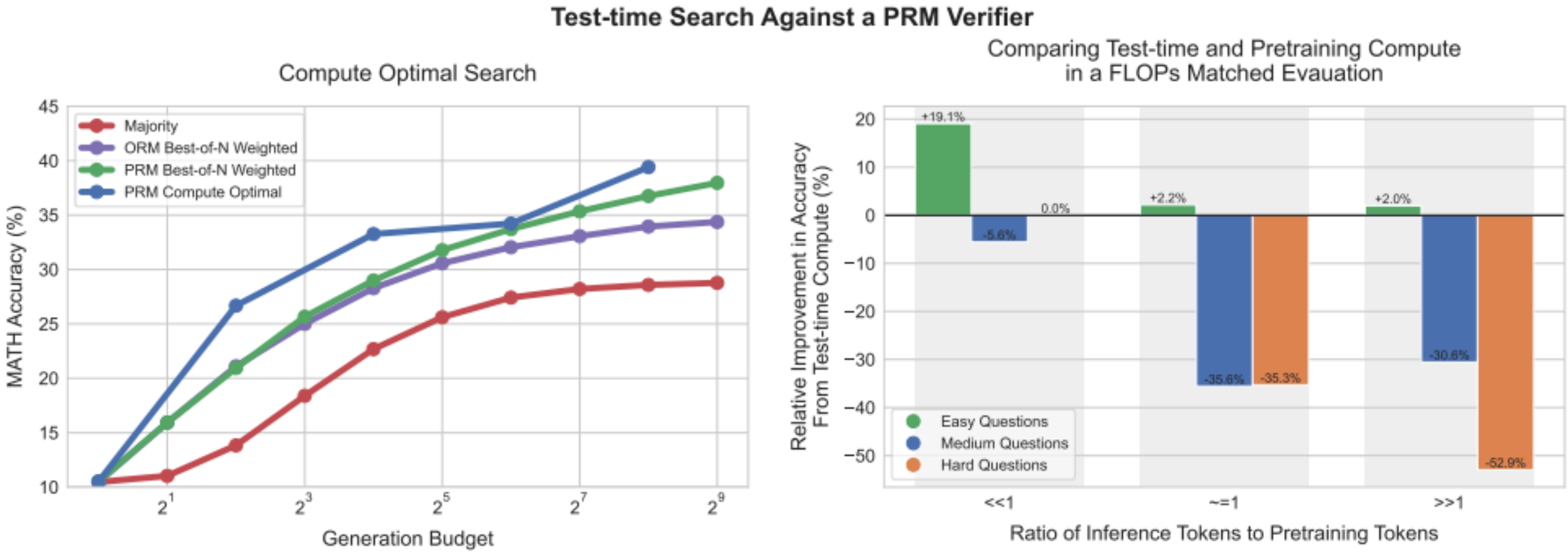
Iteratively revising answers





Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters [Snell et al. 2024] - https://arxiv.org/pdf/2408.03314

Test-time search against PRM verifier



Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters [Snell et al. 2024] - https://arxiv.org/pdf/2408.03314

Comparing search for small models

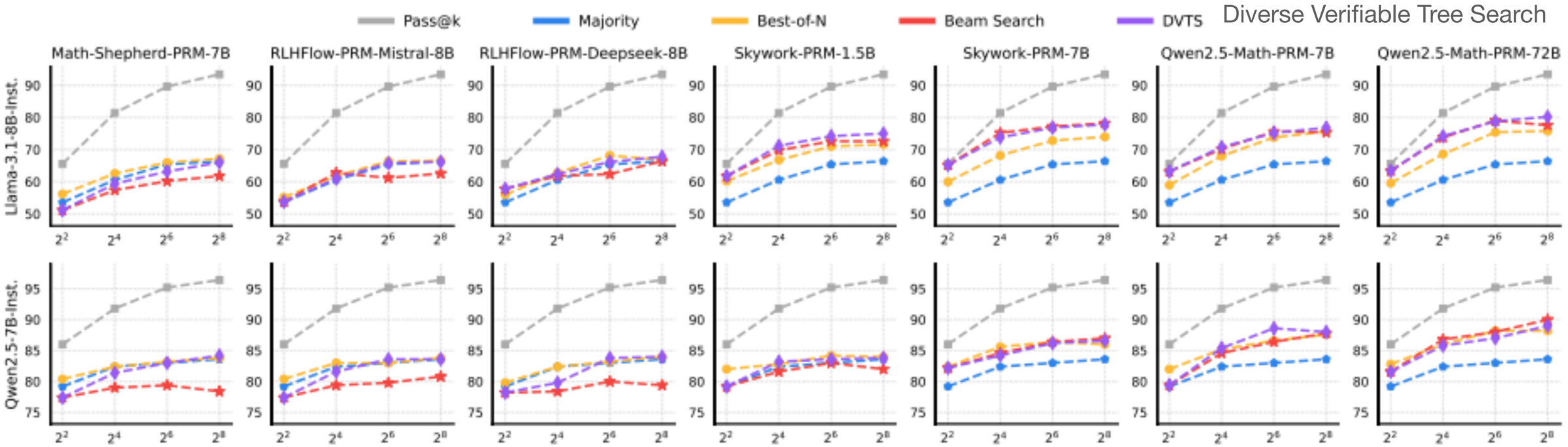
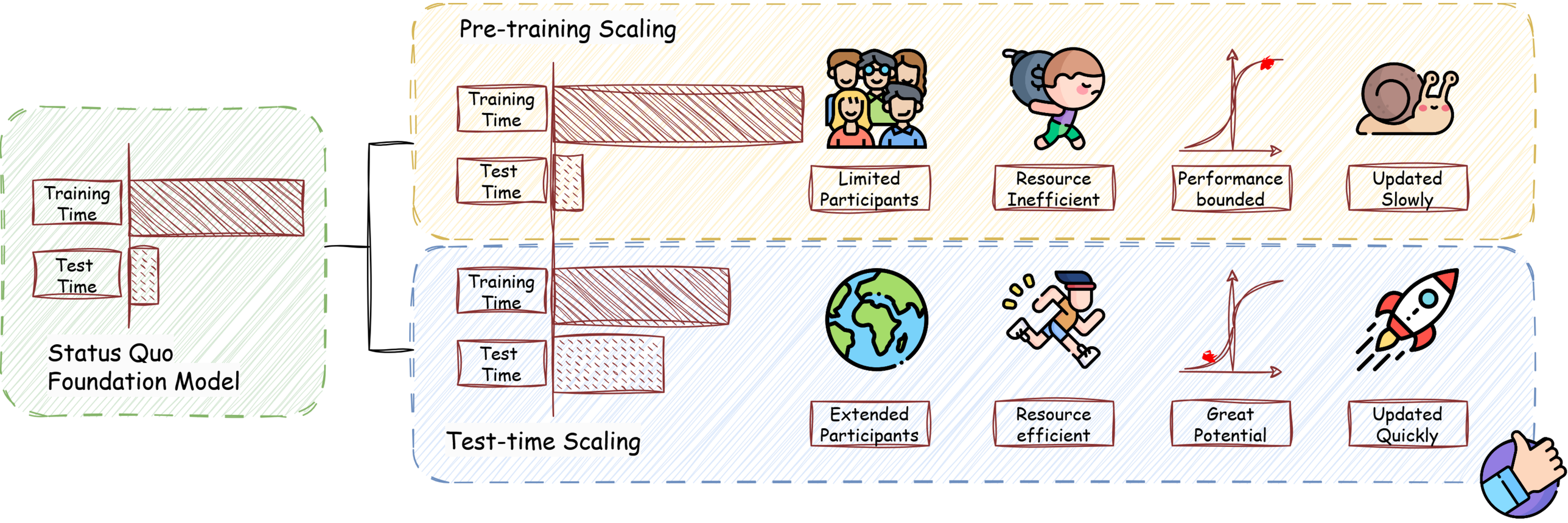


Figure 4: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on MATH-500 with different PRMs and TTS strategies.

Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling [Liu et al. 2025] - https://arxiv.org/pdf/2502.06703

Test-time scaling



What, How, Where, and How Well? A Survey on Test-Time Scaling in Large Language Models
[Zhang et al. 2025] - https://testtimescaling.github.io/