MOHAMED BIN ZAYED UNIVERSITY OF ARTIFICIAL INTELLIGENCE

BUILDING LLMS: SCIENCE, ENGINEERING, AND OPEN INNOVATION

Hector Zhengzhong Liu
Director
MBZUAI Institute of Foundation models - Silicon Valley Lab
https://mbzuai.ac.ae/institute-of-foundation-models/

THE LLM TRAINING PIPELINE

And the science and engineering

LLM Training requires a lot of budget planning and tradeoff.

What Should You Know as a Pre-Trainer

- Today, your boss knock on your door:
  - Boss: Can you to prepare an LLM training proposal by EOD? Let's beat GPT-4 next week.
  - You: ??
- How do we approach to plan these?

Large Scale Training is about Tradeoff

- Engineering decisions about making tradeoff of resources
- The science here is often about quantifying and predicting the tradeoffs

Determine Goals and Budgets

- As a training team lead, you need to first figure out the goals of budgets of your model.
  - What's the major use case of the model?
    - What's the major knowledge domains that need to be covered? Finance, bio-medical or legal problems?
    - What's the ability the model should have? Logical reasoning or programming?
  - Figure out the model use cases:
    - Evaluation and Target Scores
    - Data Choices
    - Maybe even Model Capability (e.g., Model Size, Token Size)

Example Optimization Goals

Performance per dollar

- Find a training setting to maximize performance per dollar spent
- Dollar can be spent on data, computation and other resources

Performance

(over training TPP > 20)

- Highest performance achievable given some constraints(e.g., limited data)

ROI

- In practice, the optimal compute budget requires one to consider the tradeoff between cost of pretraining and cost of inference

Key Decision Model Capacity

- Models designed with high capacity can (potentially) achieve high performance
- Factors Affecting Model Capacity
  - Model Architecture (Transformer vs. RNN vs. State Space Model)
  - Training FLOPs
    - # Model Parameters
    - # Tokens

Neural Scaling Law: studying behaviors of neural networks that are predictable with scaling training time, dataset size and model size across many orders of magnitude

Scaling Laws

Chinchilla Scaling Law Formulation

L: Loss

N: #Model Parameter

D: #Training Token

E: A constant capturing the Entropy of the text

E, A, B, α, and β are to be fit during experiments

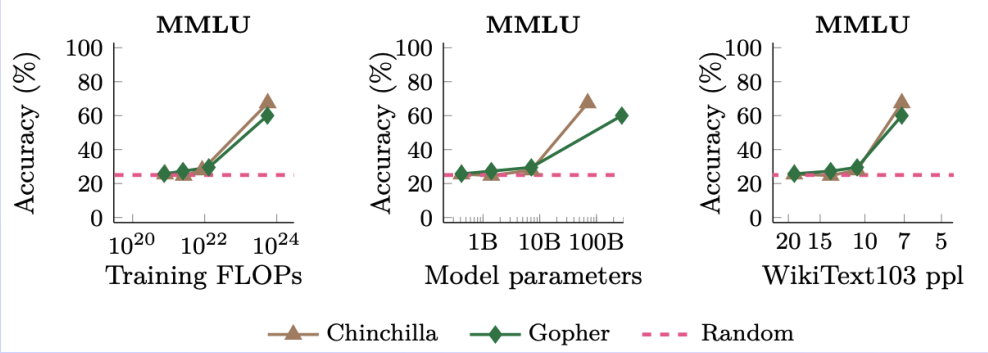
$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}.$$

Scaling law study allows one to estimate the model behaviors of high capacity by experimenting on low-capacity ones.

You will see scaling law being applied to almost all aspects for model setup

Model Capacity

- The Phenomenon of Emergent Ability makes the capacity decision more important
  - Choose the right budget that reaches desired ability



	Emergent scale			
	Train. FLOPs	Params.	Model	Reference
Few-shot prompting abilities				
• Addition/subtraction (3 digit)	2.3E+22	13B	GPT-3	Brown et al. (2020)
• Addition/subtraction (4-5 digit)	3.1E+23	175B		
• MMLU Benchmark (57 topic avg.)	3.1E+23	175B	GPT-3	Hendrycks et al. (2021a)
• Toxicity classification (CivilComments)	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Truthfulness (Truthful QA)	5.0E+23	280B		
• MMLU Benchmark (26 topics)	5.0E+23	280B		
• Grounded conceptual mappings	3.1E+23	175B	GPT-3	Patel & Pavlick (2022)
• MMLU Benchmark (30 topics)	5.0E+23	70B	Chinchilla	Hoffmann et al. (2022)
• Word in Context (WiC) benchmark	2.5E+24	540B	PaLM	Chowdhery et al. (2022)
• Many BIG-Bench tasks (see Appendix E)	Many	Many	Many	BIG-Bench (2022)
Augmented prompting abilities				
• Instruction following (finetuning)	1.3E+23	68B	FLAN	Wei et al. (2022a)
• Scratchpad: 8-digit addition (finetuning)	8.9E+19	40M	LaMDA	Nye et al. (2021)
• Using open-book knowledge for fact checking	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Chain-of-thought: Math word problems	1.3E+23	68B	LaMDA	Wei et al. (2022b)
• Chain-of-thought: StrategyQA	2.9E+23	62B	PaLM	Chowdhery et al. (2022)
• Differentiable search index	3.3E+22	11B	T5	Tay et al. (2022b)
• Self-con				
• Leverag				
• Least-t				
• Zero-sh				
• Calibra				
• Multilingual chain-of-thought reasoning	2.9E+23	62B	PaLM	Shi et al. (2022)
• Ask me anything prompting	1.4E+22	6B	EleutherAI	Arora et al. (2022)

Key metric/ability may only start to emerge (i.e., beyond random) when model exceed certain capability

List of Emergent Ability at Different Model Capacity

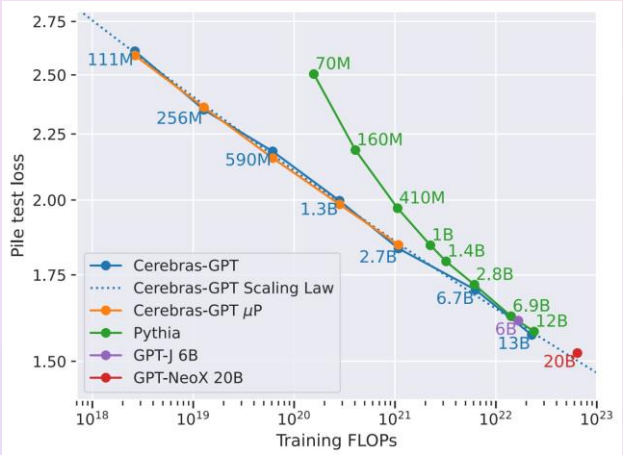
- Emergent Abilities of Large Language Models, arXiv:2206.07682

The Power of Scaling Law

- Conducting careful scaling laws help predict various model behaviors
  - In [1], a 12B model's memorized sequences can be (somewhat) predicted by smaller models

Model	Precision	Recall
Pythia-70M	0.956	0.197
Pythia-160M	0.948	0.289
Pythia-410M	0.940	0.401
Pythia-1.0B	0.931	0.512
Pythia-1.4B	0.926	0.554
Pythia-2.8B	0.909	0.658
Pythia-6.9B	0.884	0.795
Pythia-12B	—	—

Example: Text Memorization Prediction with scaling law



Scaling law of different model families [2]

[1] Emergent and Predictable Memorization in Large Language Models, arXiv:2304.11158

[2] Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster, arXiv:2304.03208

Extra Tradeoffs

- Research/Preparation Cost vs. Training Cost
  - The cost to fit a good scaling law curve is also significant, considering all the factors to be tried (e.g., hyperparameters, data selection)
  - Though suboptimal, one set of parameter may be reused when training conditions are similar (similar domain)
- Training Resource vs. Supporting Resource
  - Don’t use all your GPUs for training, always reserve enough for evaluation, analysis
  - Tradeoff between model evaluation frequency vs. rollback cost

Details of Scaling Law

- During implementation, we find there are many details to control for scaling law study
  - E.g., some noise in smaller scale training can cause the scaling law to be unstable.
  - Our team are gathering on more details.

Data is probably one of the most important step for LLM pretraining

Data Collection

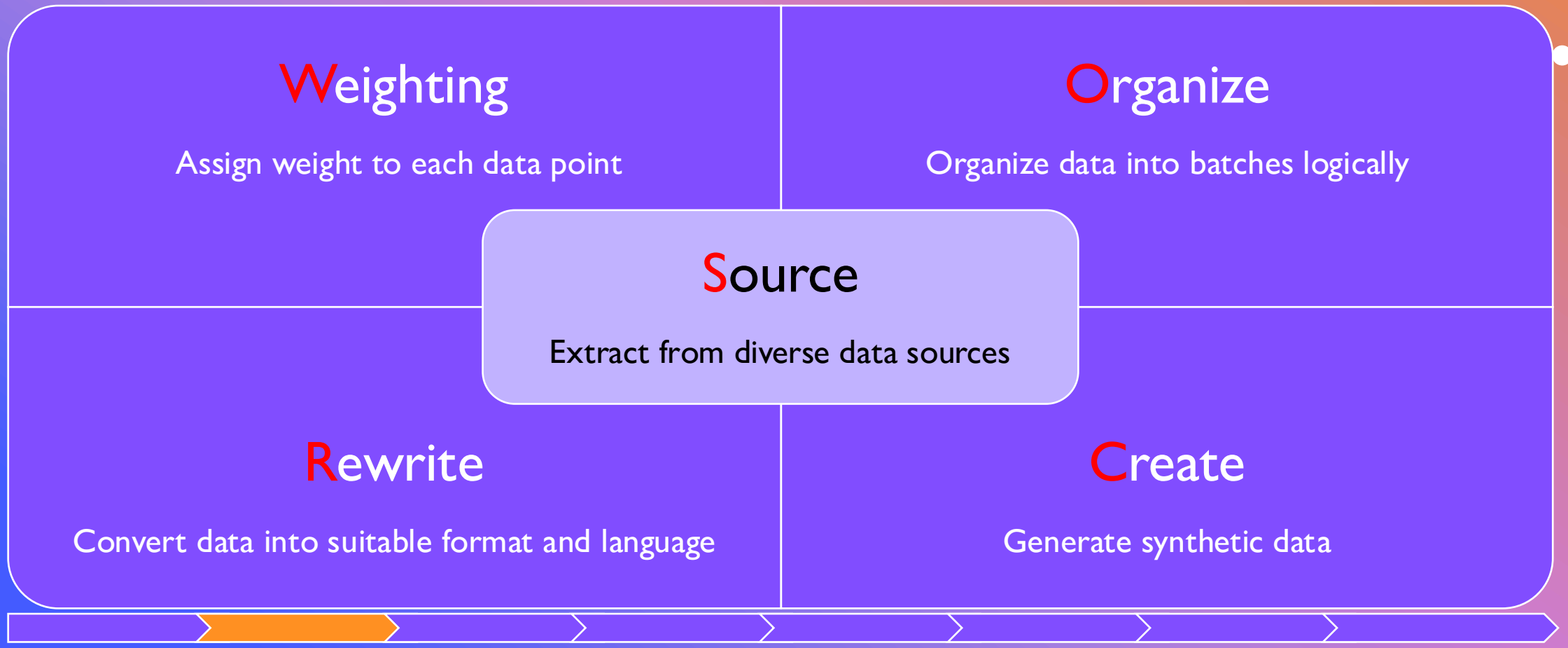
- Recall in Scaling Law, data plays a crucial role in final performance
- Collect high quality and large corpus is essential in producing in the final model
  - Data size determine D
  - Data quality changes B and β

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \boxed{\frac{B}{D^{\beta}}}.$$

Data Size captured by D, and data quality captured by B and β

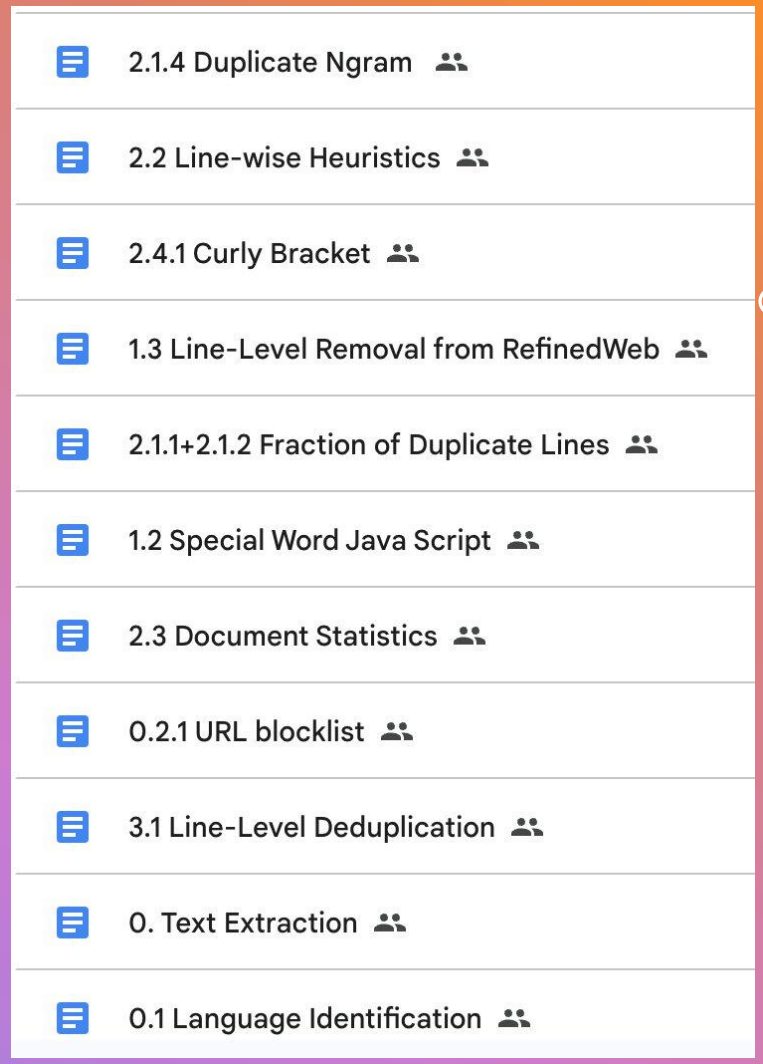
- Determine the data size based on budget and loss goal

The WORCS Framework



Heuristic Filtering

- To deal with Internet scale data, the typical way is to filter documents based on heuristic rules
  - Weighting: remove lowest quality documents (set their weight to 0)
  - Rewrite: remove certain lines (e.g., website boilerplate) from documents)



The filtering steps of TxT360, with fully documented decision process

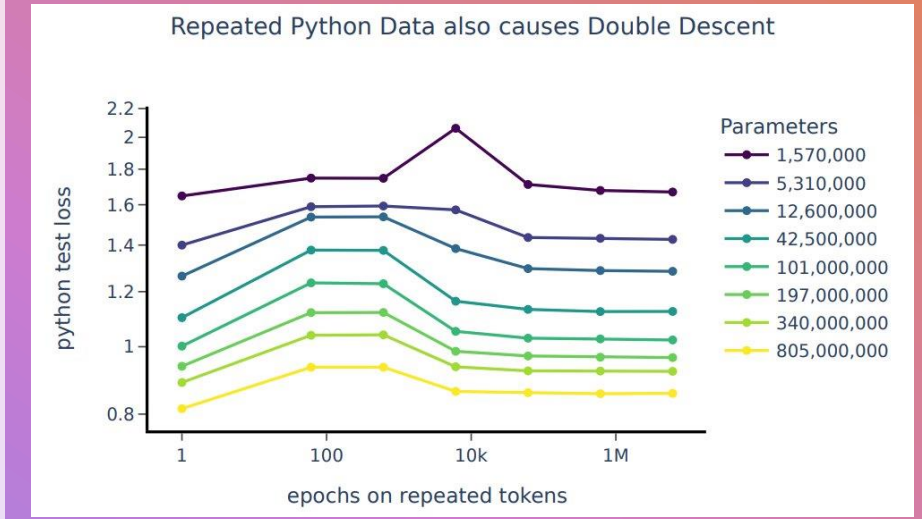
Weighting: Controlling the amount of duplication

- Empirically, most study confirm that deduplication can help improve model quality [1,2,3]
  - Falcon uses Refinedweb
  - LLM360/Crystal uses SlimPajama
- There is also attempts to repeat training data
  - [4] train the model on multiple epochs
  - Llama1 and several of our models (LLM360/Amber, K2) also upweight certain high quality dataset (such as Wikipedia). K2 performs pretty well after upweighting.
- For full weighting control, keep unique documents only

[1] To Repeat or Not To Repeat: Insights from Scaling LLM under Token-Crisis, arXiv:2305.13230
[2] Deduplicating Training Data Makes Language Models Better, arXiv:2107.06499
[3] Scaling Laws and Interpretability of Learning from Repeated Data, arXiv:2205.10487
[4] Scaling Data-Constrained Language Models, arXiv:2305.16264

Deduplication

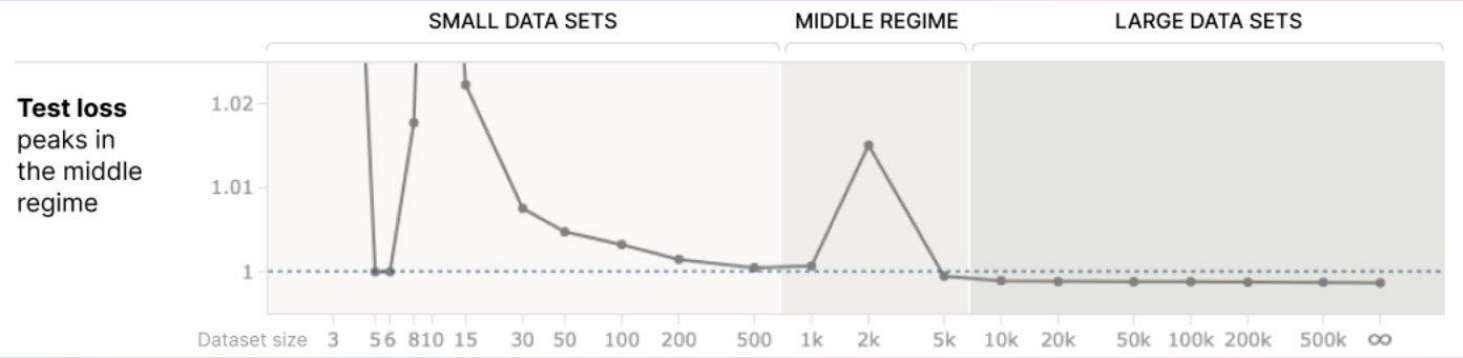
- [1] hypothesize that duplicate data would cause the model to replace generalization ability with memorization
  - Duplicate data induce a double-descent phenomenon
    - Repeat a few times does not cause much damage
    - Repeat very many times also does not cause much damage
    - Eventually grokking would happen hard to predict in real-world scenarios



[1] Scaling Laws and Interpretability of Learning from Repeated Data, arXiv:2205.10487
[2] Superposition, Memorization, and Double Descent, Transformer Circuits Thread

Deduplication

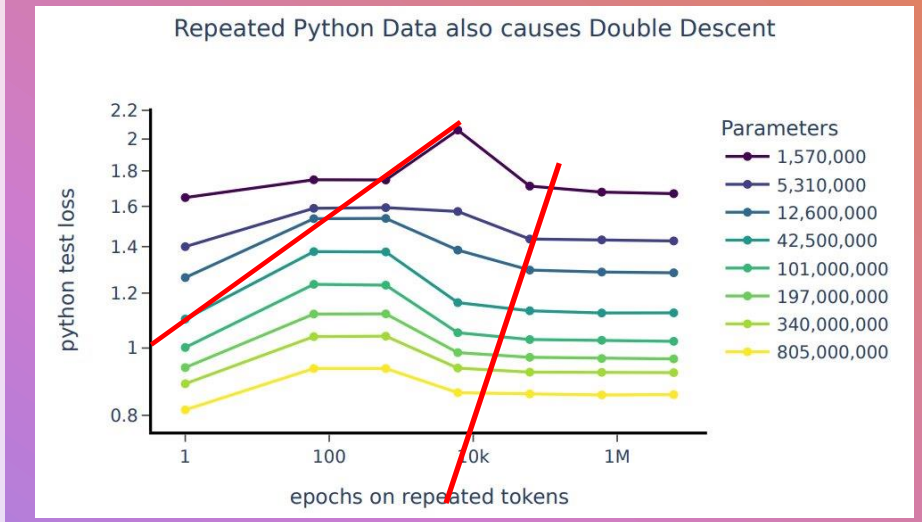
- Connection between this double-descent phenomenon with superposition
  - Superposition: a model can represent more features than the dimension it has
  - [2] shows a double-descent phase change related to superposition and memorization



[1] Scaling Laws and Interpretability of Learning from Repeated Data, arXiv:2205.10487
[2] Superposition, Memorization, and Double Descent, the Transformer Circuits Thread

Deduplication

- Deduplicating your dataset is always good, even if you want to duplicate the data later, you can control which portion to duplicate precisely
  - In [2], the authors observed gains with 4 epochs of repetition similar to unique data, but diminishing return
  - However, the study of [2] is done on maximally 9B model, notice that the double descent penalty comes early with larger models
    - Empirically, we find that performance a 65B model seems fine from 3-4 times of repetition

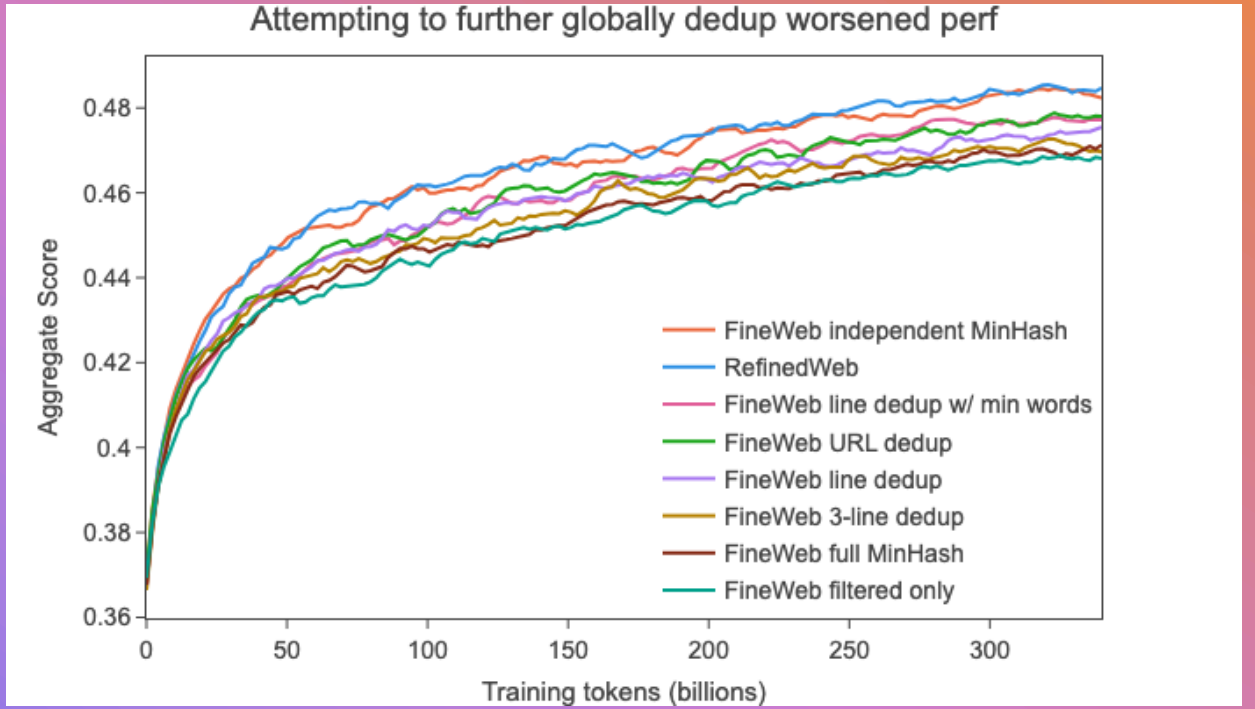


The region of double descent comes earlier with larger models

[1] Emergent and Predictable Memorization in Large Language Models, arXiv:2304.11158
[2] Scaling Laws and Interpretability of Learning from Repeated Data, arXiv:2205.10487
[3] Scaling Data-Constrained Language Models, arXiv:2305.16264

Deduplication

- The Fineweb [1] report shows that deduplicating more greedily and globally actually hurts performance.
- Deduplicating within each dump (organized by year) is the best?!
  - [1] suggests if a doc does not have duplications across dumps (years), they are low quality
  - More -> downsample high quality data



[1] https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

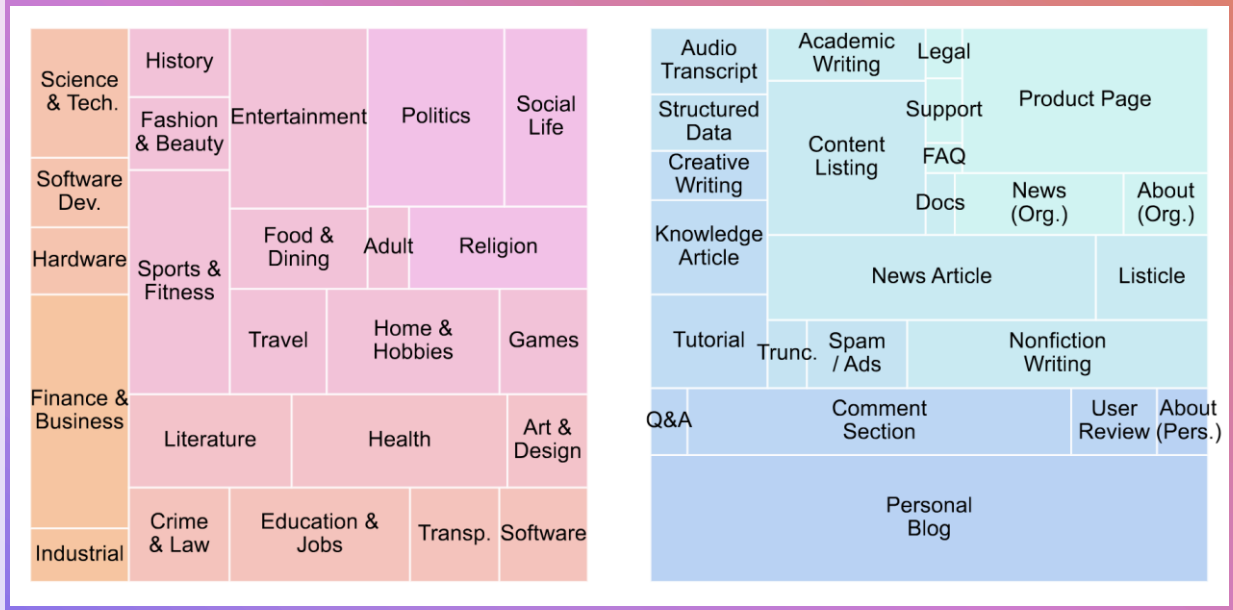
Deduplication

- Take-away:
  - Frequent duplication has a negative impact on the performance have mixed empirical results.
  - Deduplication is just another form for data weighting
    - Know exactly how much you sampled
    - Know what you remove is important
  - Monitor when the training process reaches repeated data
    - Check for key metrics at the start of epochs
    - Frequently sample generation results for qualitative checks
    - Evaluate the gap between empirical vs. predicted validation loss

[1] Emergent and Predictable Memorization in Large Language Models, arXiv:2304.11158
[2] Scaling Laws and Interpretability of Learning from Repeated Data, arXiv:2205.10487
[3] Scaling Data-Constrained Language Models, arXiv:2305.16264

Data Domains

- Pretraining datasets are often curated via various sources, which creates natural data domains.
- The major data source, web, can be further clustered into sources.
  - Tailored rules/methods can be applied to different sources.

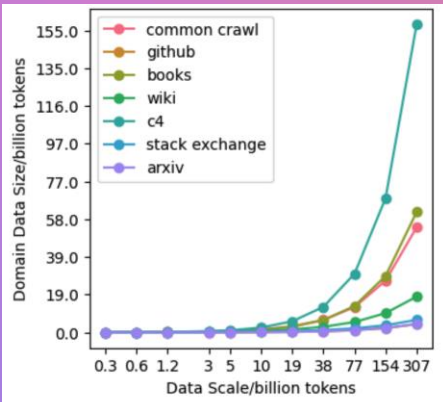


[1]
[2] *Organize the Web: Constructing Domains Enhances Pre-Training Data Curation*

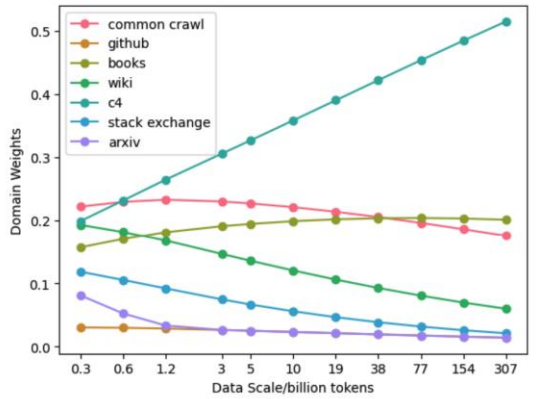
Domain Weighting

- Weighting is also often done on domain/cluster level
  - Fine-grained document level is theoretically sound but practically inefficient.
  - Studies [1,2] show that proper data weighting can impact the performance significantly
- Domain weighting can be considered as a mixture of of multiple data scaling law
- Several sub-problems:
  - Clustering and domain classification
  - Regression, meta learning of domain weights
  - Scaling law and transferability of the weights

Benchmark	Worst Model	Best Model	Δ
Social IQA (Sap et al., 2019)	32.4	33.9	1.5
HellaSwag (Zellers et al., 2019)	33.0	43.4	10.4
PiQA (Bisk et al., 2020)	60.2	69.0	8.8
OpenBookQA (Mihaylov et al., 2018)	25.8	31.2	5.4
Lambada (Paperno et al., 2016)	18.9	33.5	14.6
SciQ (Welbl et al., 2017)	76.7	82.9	6.2
ARC Easy (Clark et al., 2018)	44.9	52.2	7.3
COPA (Sarlin et al., 2020)	61.5	70.5	9.0
RACE (Lai et al., 2017)	27.9	32.5	4.6
LogiQA (Liu et al., 2020)	23.2	27.7	4.5
QQP (Wang et al., 2018)	48.0	59.7	11.7
WinoGrande (Sakaguchi et al., 2021)	50.3	53.2	2.9
MultiRC (Khashabi et al., 2018)	47.6	55.7	8.1
Average Performance	43.7	47.9	4.2



(a) AutoScale -predicted optimal data quantity for each domain as training data scales up.



(b) AutoScale -predicted optimal domain weights as training data scales up.

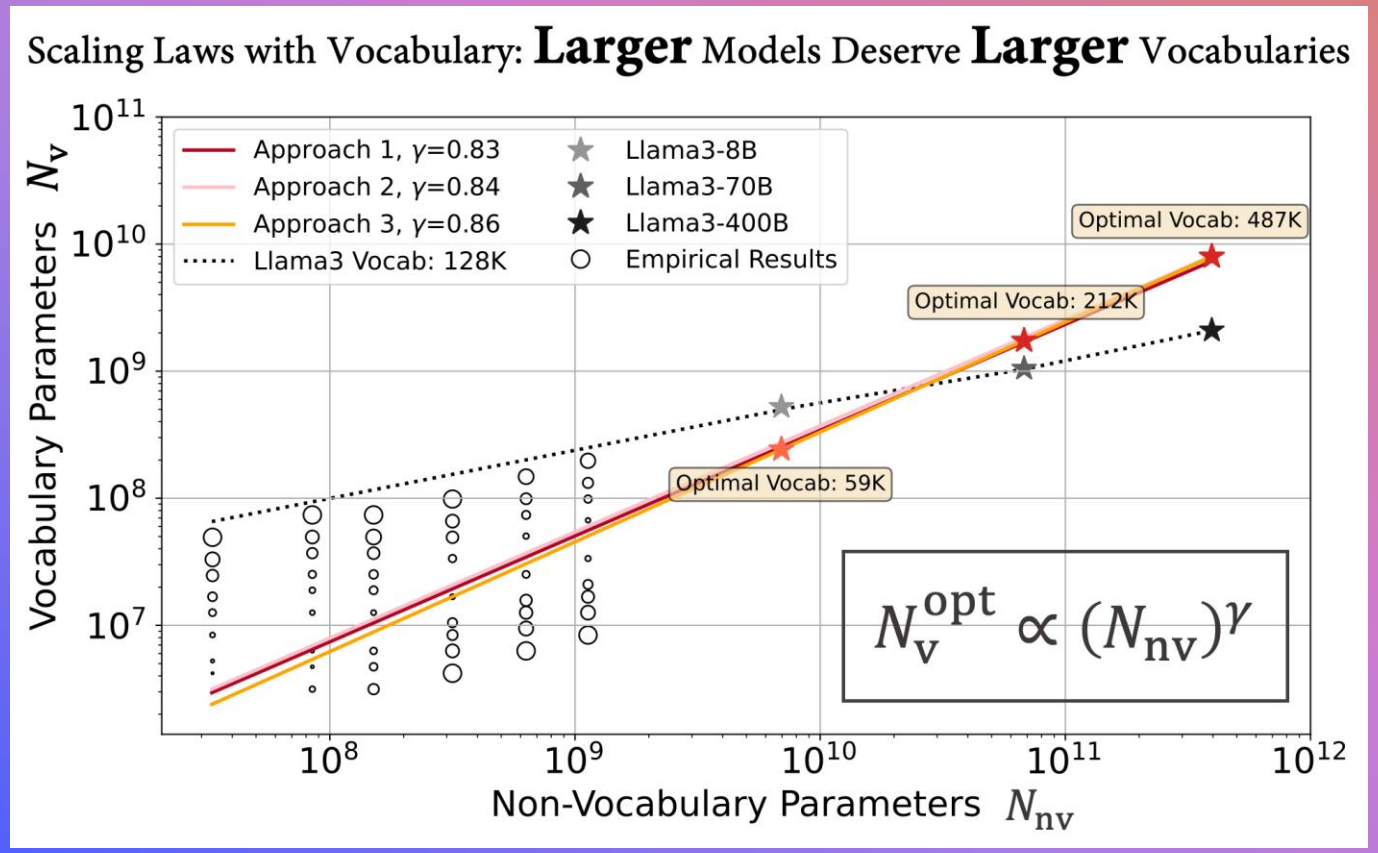
[1] REGMIX: DATA MIXTURE AS REGRESSION FOR LANGUAGE MODEL PRE-TRAINING
[2] AutoScale: Scale-Aware Data Mixing for Pre-Training LLMs

Vocabulary

- Vocabulary is typically determined by a subword tokenization algorithm over the dataset.
- A few important decisions:
  - Special characters (such as control tokens in StarCoder)
  - Vocabulary size: a hyperparameter to choose
  - If multilingual, take special care with tokenization [1], especially with the case of continuously pretraining
- A few metrics can be used to help decisions:
  - Fertility
  - # of unseen (sub)word

[1] How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. arXiv:2012.15613
[2] StarCoder: may the source be with you! arXiv:2305.06161

Vocabulary



[1] Scaling Laws with Vocabulary: Larger Models Deserve Larger Vocabularies. arxiv:2407.13623

TxT360

- TxT360: a dataset of both Web data + Non Web curated data with Global dedup
  - This allows one to have precise control of which data point to use, enabling precise weight control

Our simple weighting recipe shows a better learning curve than baselines. Read more in our blog [1]

[1] https://huggingface.co/spaces/LLM360/TxT360

TxT360 Blog: the tedious but useful documentation

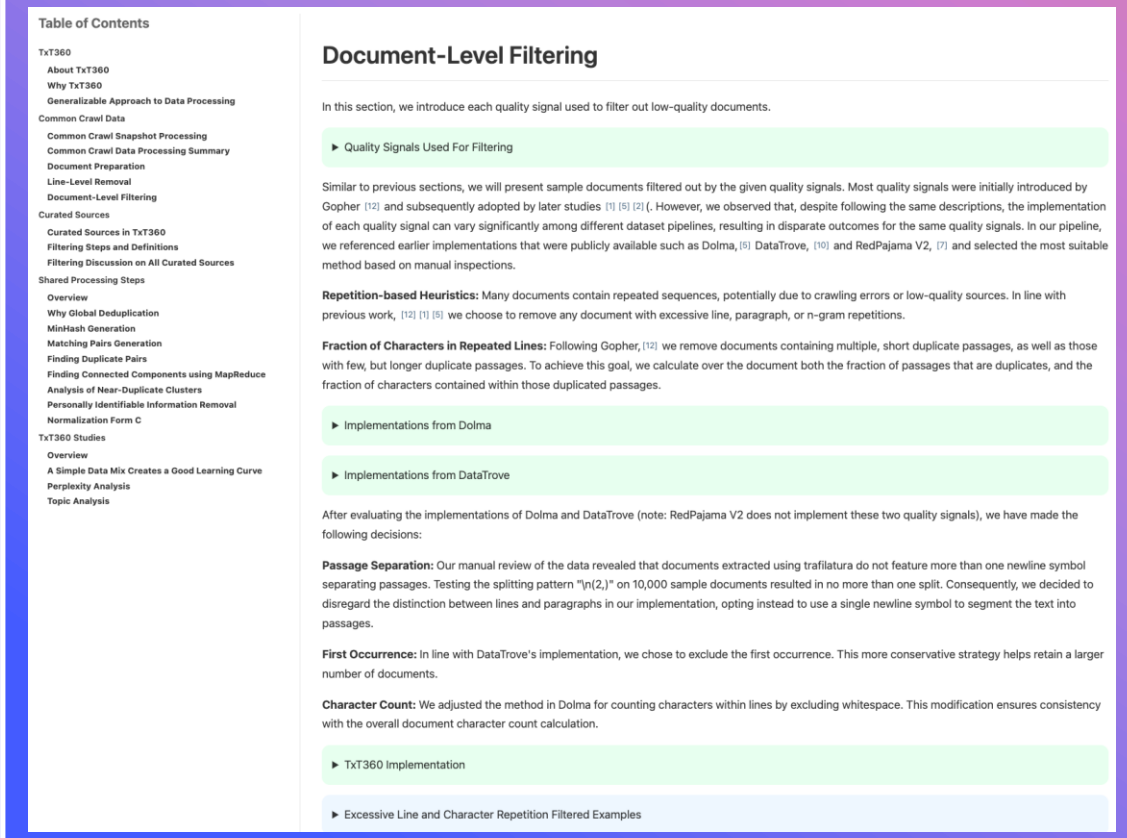


Table of Contents

Document-Level Filtering

In this section, we introduce each quality signal used to filter out low-quality documents.

▸ Quality Signals Used For Filtering

Similar to previous sections, we will present sample documents filtered out by the given quality signals. Most quality signals were initially introduced by Gopher [12] and subsequently adopted by later studies [1] [5] [2] (. However, we observed that, despite following the same descriptions, the implementation of each quality signal can vary significantly among different dataset pipelines, resulting in disparate outcomes for the same quality signals. In our pipeline, we referenced earlier implementations that were publicly available such as Dolma, [5] DataTrove, [10] and RedPajama V2, [7] and selected the most suitable method based on manual inspections.

Repetition-based Heuristics: Many documents contain repeated sequences, potentially due to crawling errors or low-quality sources. In line with previous work, [12] [1] [5] we choose to remove any document with excessive line, paragraph, or n-gram repetitions.

Fraction of Characters in Repeated Lines: Following Gopher, [12] we remove documents containing multiple, short duplicate passages, as well as those with few, but longer duplicate passages. To achieve this goal, we calculate over the document both the fraction of passages that are duplicates, and the fraction of characters contained within those duplicated passages.

▸ Implementations from Dolma

▸ Implementations from DataTrove

After evaluating the implementations of Dolma and DataTrove (note: RedPajama V2 does not implement these two quality signals), we have made the following decisions:

Passage Separation: Our manual review of the data revealed that documents extracted using trafilatura do not feature more than one newline symbol separating passages. Testing the splitting pattern "\n{2,}" on 10,000 sample documents resulted in no more than one split. Consequently, we decided to disregard the distinction between lines and paragraphs in our implementation, opting instead to use a single newline symbol to segment the text into passages.

First Occurrence: In line with DataTrove's implementation, we chose to exclude the first occurrence. This more conservative strategy helps retain a larger number of documents.

Character Count: We adjusted the method in Dolma for counting characters within lines by excluding whitespace. This modification ensures consistency with the overall document character count calculation.

▸ TxT360 Implementation

▸ Excessive Line and Character Repetition Filtered Examples

- We release a very detailed blog describing every steps of the implementation, including comparison and data samples
  - It is tedious to read but you will find everything you need to reproduce
  - A technical paper with more results is coming soon.

[1] https://huggingface.co/spaces/LLM360/TxT360

Choose
your architecture
that fits your
requirements

Model Architecture Choices

- There are now quite a few architectures
  - Transformer based: GPT series, Llama variants
  - State Space Model: Mamba, Striped Hyena
  - RNN like models: RWKV
- In LLM360 experiments, transformer-based models still work consistently well.
  - In our preliminary study, we find SSMs are hard to be trained for coding tasks
- To make training and inference work well, some model choices can be hardware dependent

Hardware Aware Decisions

- The actual engineering process depends a lot on the underlying hardware.
  - E.g., on Nvidia GPUs we would want to control matrix dimensions to be multipliers of 8 or 16 according to the official documentation[1]
  - More parameter suggestions can be found in a study [2], key heuristic in the table

Parameter	Recommendation
Vocab size	divisible by 64
b	As large as possible
b*s, h/a, and h/t	divisible by a power of 2
(b*a)/t	should be an integer
t	As small as possible

a: # attention heads
s: seq length
t: tensor-parallel size
h: hidden dimension size
b: microbatch size

[1] https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html#requirements-tc
[2] The Case for Co-Designing Model Architectures with Hardware, arXiv:2401.14489

Model Architecture Efficiency

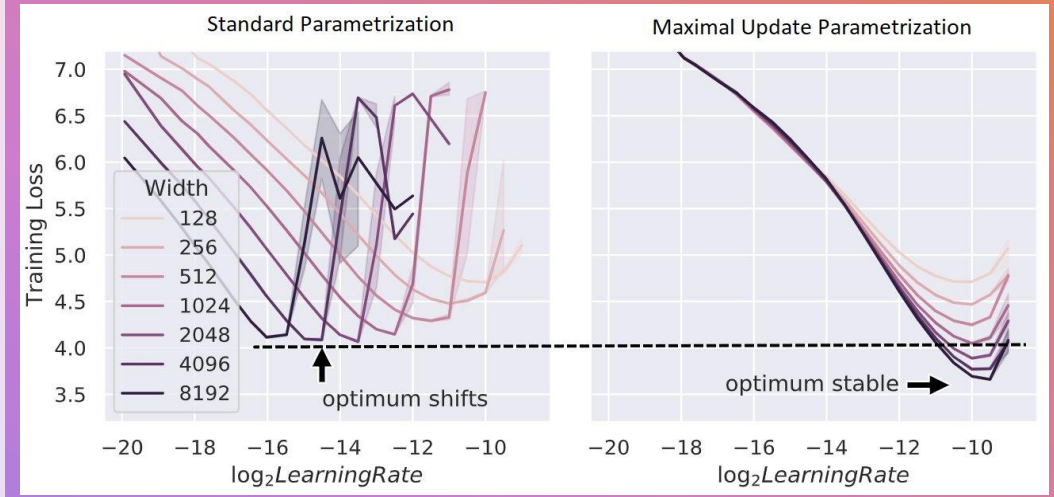
- Bottlenecks in LLM models often happen with memory constraints
  - KV-Cache Memory
  - Group Query Attention in place of regular MQA
  - Flashattention [1]
- Architecture design often need to consider the efficiencies of an architecture on hardware
  - Mamba [2] has better FLOPs but seems to be slower according to Wall Time

[1] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, Dao et.al. 2022
[2] Mamba: Linear-Time Sequence Modeling with Selective State Spaces, Dao etl.al. 2023

Tuning hyperparameters makes a difference in a long term

Hyperparameter Study

- Goal: you want your model to behave nicely over the entire training run
  - Training samples contribute equally to your model
  - Gradient has fewer noise and on the right direction
- Often conducted during scaling law study
  - How do we know if some hyperparameter can work across model sizes?
  - Some tools are available for us, particularly μP and μTransfer [1]



Under μP settings, [1] shows that log2(LearningRate) is stable across different model width

[1] Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer, arXiv:2203.03466

µTransfer

Algorithm 1 Tuning a Large Target Model via μTransfer

1: Parametrize target model in Maximal Update Parametrization (μP)
2: Tune a smaller version (in width and/or depth) of target model
3: Copy tuned hyperparameters to target model

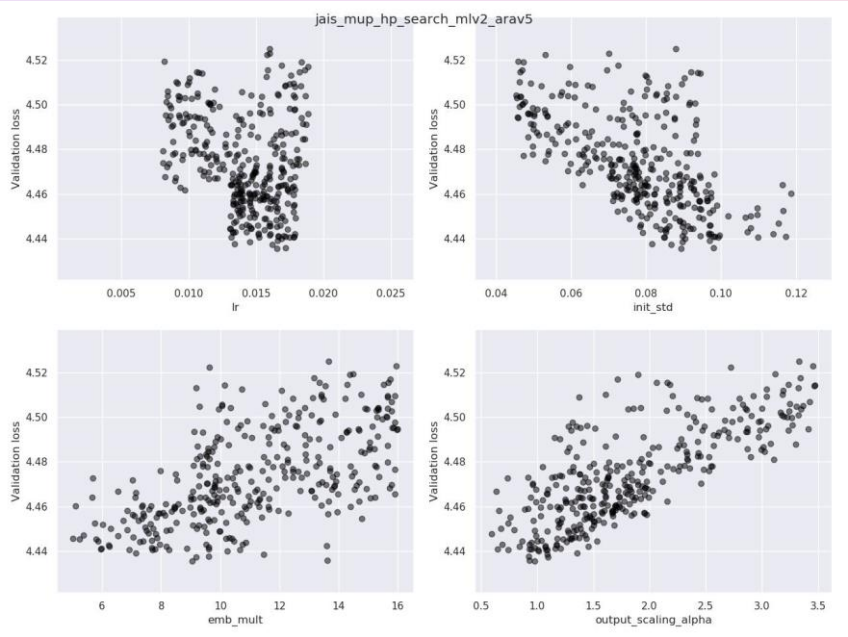
- µP advantage:
  - Transferring model parameters tuned in µP is very scalable across model size*
  - Empirically, weight norm and gradient norms behave similar across scale
    - Can use this as a sanity check for training implementation)
- Zero-shot transfer hyperparameters via µTransfer

μTransferable	Not μTransferable	μTransferred *Across*
optimization related, init, parameter multipliers, etc	regularization (dropout, weight decay, etc)	width, depth*, batch size*, training time*, seq length*

- * In Tensor Program V, µP only proves the mechanism for “widthwise hyperparameter transfer”, though people use the method beyond the theoretical guarantee (applied on depth, different data sizes).
- Recently in Tensor Program VI, “depthwise transfer” is proposed:
- Tensor Programs VI: Feature Learning in Infinite-Depth Neural Networks, arXiv:2310.02244

Hyperparameter Study: Tune on the Proxy

- We can first tune hyperparameter efficiently on a small model (the proxy model)



A hyperparameter search experiment when training JAIS

μTransfer

- μTransferable parameters can be zero-shot transfer from the proxy model to large model (small -> large), which works very well with Scaling Law

Parameter	Usage	Data Type	Value
model.scale_qk_dot_by_d	$(Q^T K/d_{head})$V	Bool	True
model.output_logits_scale	$Y_{logits} = W_{unemb}X/\tilde{d}_{model}$	Float	$1/\tilde{d}_{model}$
model.embeddings_scale	$Y_{embd} = m_{emb}.\,embed(X)$	Float	m_{emb} (tunable)
optimizer.adjust_learning_rate	$\eta_{base}/\tilde{d}_{model}$	Dict	'decoder_kernel': $1/\tilde{d}_{model}$
optimizer.embedding_initializer	$W_{emb} \sim N_{trunc}(0, \sigma^2_{base})$	Dict	'std': σ^2_{base}
optimizer.initializer	$W_{emb} \sim N_{trunc}(0, \sigma^2_{base})$	Dict	'std': σ^2_{base}
optimizer.initializer	$W_{qkv} \sim N_{trunc}(0, \sigma^2_{base}/\tilde{d}_{model})$	Dict	'std': $\sigma^2_{base}/\tilde{d}_{model}$
optimizer.output_initializer	$W_o, W_{FFN2} \sim N_{trunc}(0, \sigma^2_{base}/(2.\tilde{d}_{model}.\,n_{layers}))$	Dict	'std': $\sigma^2_{base}/(2.\tilde{d}_{model}.\,n_{layers})$

$d_{model,0}$	Proxy model's width
d_{model}	Model width
$\tilde{d}_{model}$	Width multiplier (d_{model} / $d_{model,0}$)

Tensors	Initializer	Learning Rate	Output
Embeddings	$N_{trunc}(0, \sigma^2_{base})$	η_{base}	$m_{emb}.\,embed(X)$
LN	$W^{LN} \sim 1$, $b^{LN} \sim 0$	η_{base}	-
Bias	b~0	η_{base}	-
Attention Logits	-	-	$(Q^T K/d_{head})$V
QKV Weights	$N_{trunc}(0, \sigma^2_{base}/\tilde{d}_{model})$	$\eta_{base}/\tilde{d}_{model}$	-
Attention Output Weights	$N_{trunc}(0, \sigma^2_{base}/(2.\tilde{d}_{model}.\,n_{layers}))$	$\eta_{base}/\tilde{d}_{model}$	-
FFN1 Weights	$N_{trunc}(0, \sigma^2_{base}/\tilde{d}_{model})$	$\eta_{base}/\tilde{d}_{model}$	-
FFN2 Weights	$N_{trunc}(0, \sigma^2_{base}/(2.\tilde{d}_{model}.\,n_{layers}))$	$\eta_{base}/\tilde{d}_{model}$	-
Output Logits	-	-	$W_{unemb}X/\tilde{d}_{model}$

Tables denote the μTransfer rules

Additional Notes on μP

- Besides μTransfer, μP allows each layer to separate and theoretically better learning rates
  - In μP, the learning rate of the Transformer backbone is scaled down with width while that of the embedding layer and output layer remains high.
- While using μP, we empirically find linear learning rate decay scheduling works better than cosine learning rate decay
- A few model choices are not theoretically μTransferrable (such as decay type), empirically we find them works well across scale
  - LLM360 is conducting more experiments around μP, especially with depth μP.

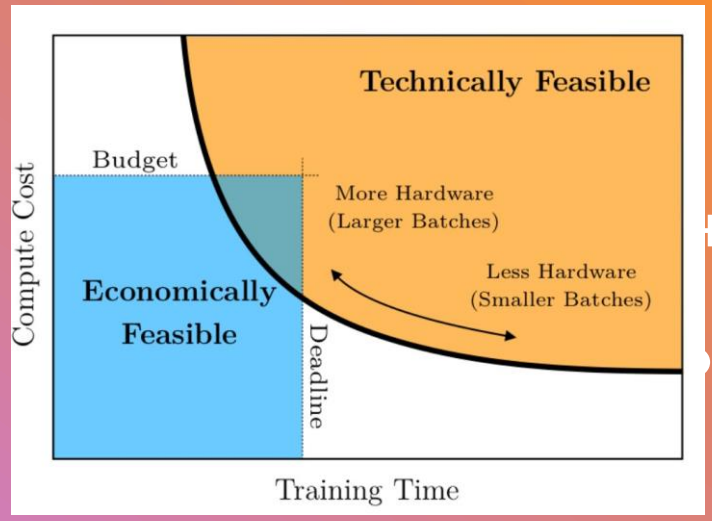
Batch Size

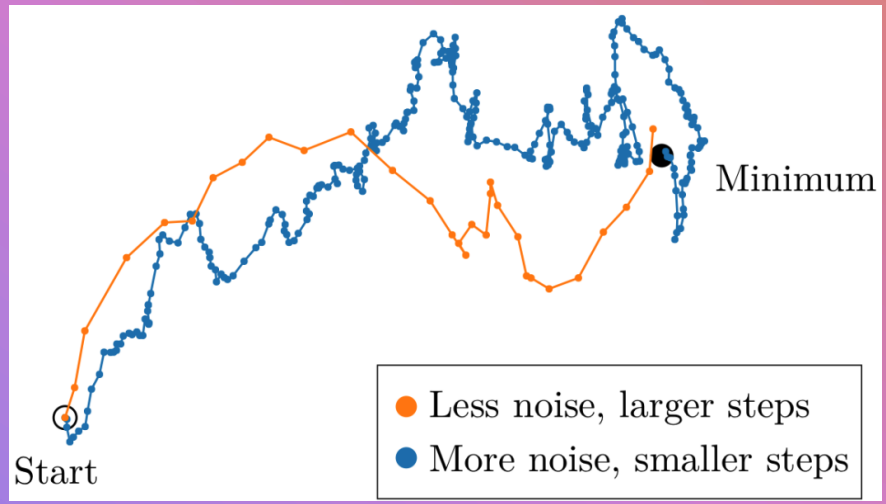
- Many LLM training uses batch size based on prior work
  - LLM360/Amber also follows Llama's 4M token batch size (2048 instance * 2048 context)
- In LLM training, the batch size need to be understood on both #instances and padded length of each instances (i.e., context size)
  - Roughly speaking, ↑ context length means ↓ number of instances packed
  - Though larger effective batch size can be achieved via gradient accumulation

[1] An Empirical Model of Large-Batch Training, arXiv:1812.06162

Batch Size

- Batch size: what is it?
  - During training, one cannot compute the real loss of the whole dataset
  - We choose a batch, and hope to use the batch loss to approximate the real loss
- What's the tradeoffs?
  - Increasing batch size will reduce the variant of loss approximation
    - Smaller batch means the gradient has more noise.
  - The return of this is diminishing
    - Once our approximation of loss is quite accurate, increasing the batch size will help very little
    - But the cost will increase for using large batches
      - The gradient update is similar, but the computation increases
  - Should identify the point where the return is not worth increasing the batch size anymore
    - Known as the Critical Batch Size [1]



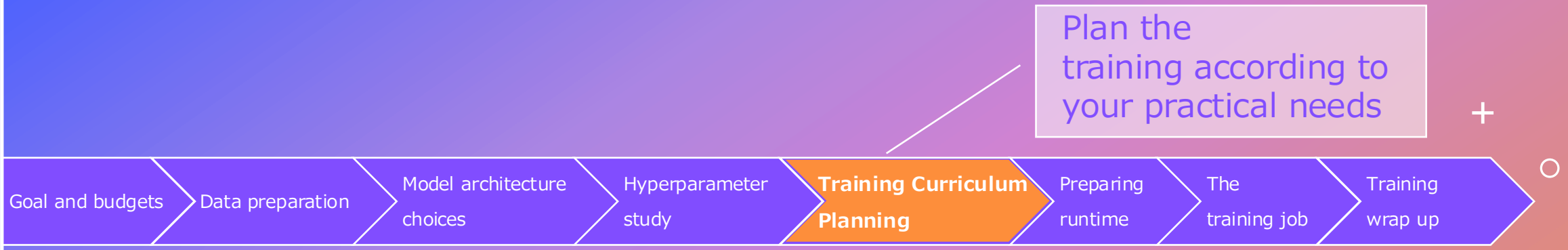


[1] An Empirical Model of Large-Batch Training, arXiv:1812.06162

Batch Size

- [1] shows the two regime of batch size
  - Small batch regime, increase batch size almost linearly improve training time
  - Large batch regime, increase batch size almost has no effect
- Gradient noise scale
  - a measure of the signal-to-noise ratio of gradient
  - When batch size is much smaller than the noise scale == small batch regime
  - When batch size is much larger than the noise scale == large batch regime
- We can find the critical batch size where the effectiveness of larger batch start to drop
  - [1] shows that this can be predicted at the order of magnitude level

[1] An Empirical Model of Large-Batch Training, arXiv:1812.06162


Plan the
training according to
your practical needs
Goal and budgets
Data preparation
Model architecture
choices
Hyperparameter
study
Training Curriculum
Planning
Preparing
runtime
The
training job
Training
wrap up

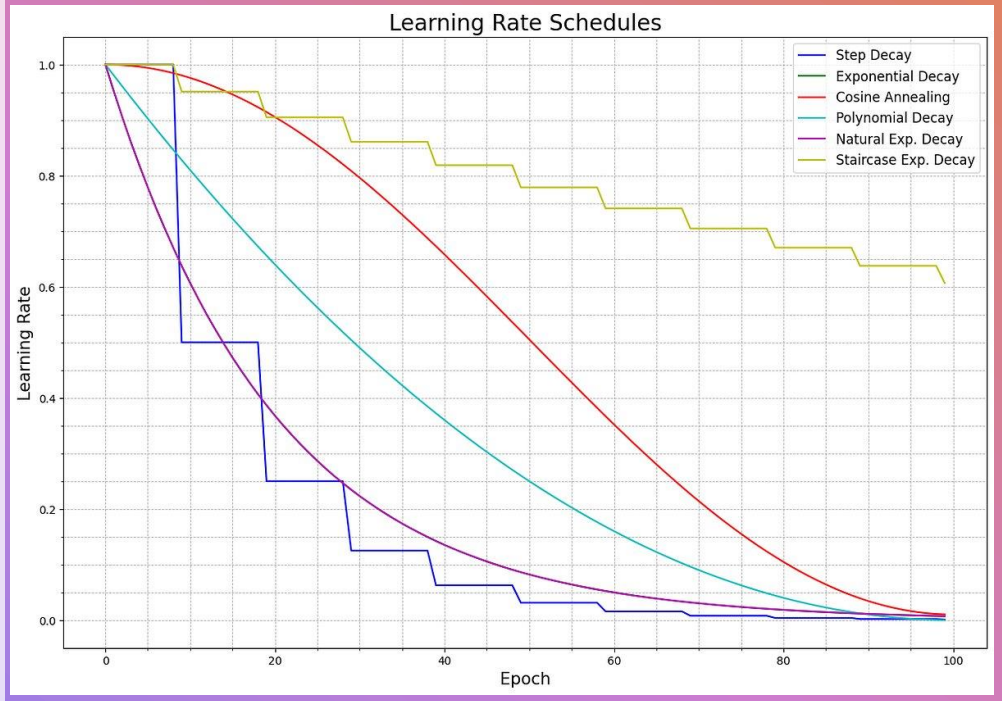
Determine Training Curriculum

- Model training is a very long and costly process,
  - Before starting, we should plan on a few key decisions.
  - Determine what happens during training, or what's the Training Curriculum*
- What should we consider in a training curriculum? For example:
  - During early stage, we hope to the model to warm-up well
  - Based on project needs, at the middle stages we could consider some special curriculum such as Model Off Ramp, or Multi Stage Training
  - During model wrap up, we want to make sure the model converge well

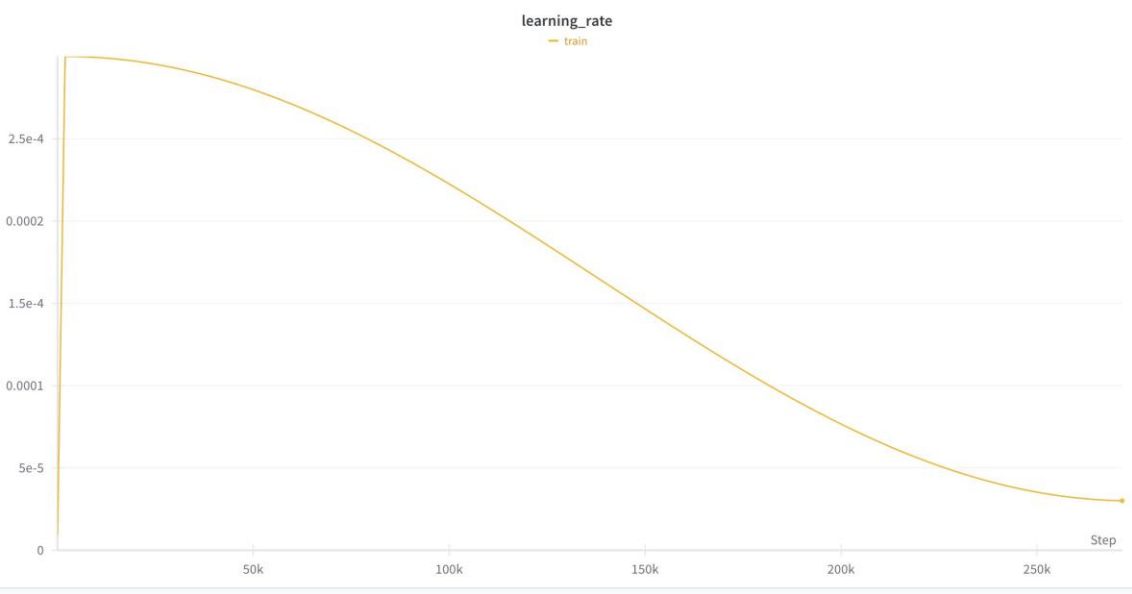
* Not to be confused with curriculum learning, though some ideas are very similar

Learning Rate Schedule

- Learning Rate Scheduling is a key choice in training LLMs
  - Since the model will be trained for months, too large or small learning rate will cause instability or major performance slow down.



Learning Rate Schedule



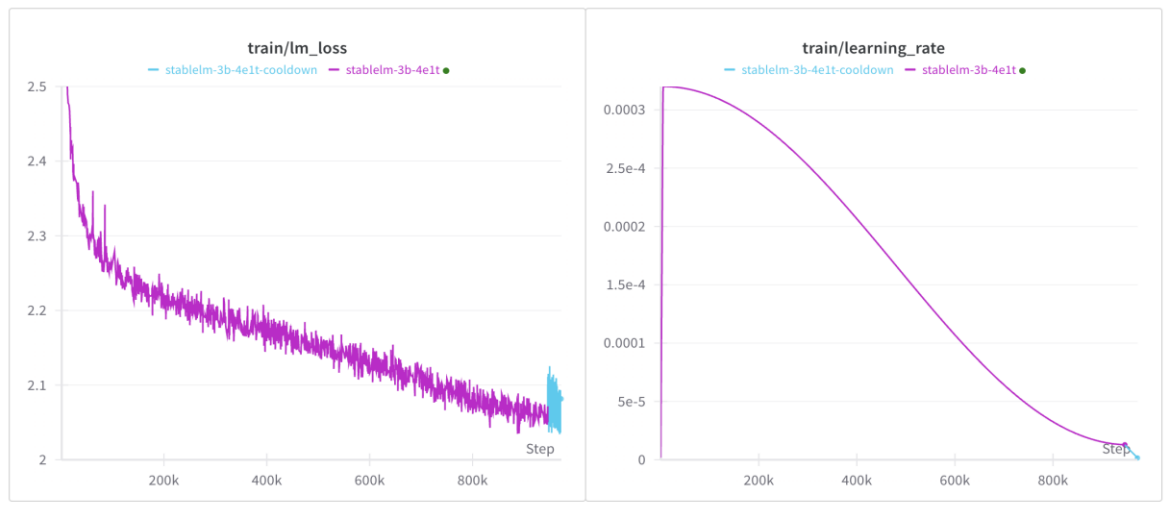
	Phase 1	Phase 2	Phase 3
LR Warmup Steps	86	86	276
LR Start Value	0.012	0.0087825	0.002
LR Final Value	0.00012408	0.00013679	0.0002
LR Decay	Linear	Linear	Linear

LLM360/Amber uses cosine decay with an initial from 3e−4 and decay to the final rate of 3e-5. The learning rate is warm up for 2,000 steps.

LLM360/Crystal's multi-phase schedule. Note that only the base LR is shown, per-layer learning is scaled with model width using μP.

In LLM360, note that Amber uses SP while Crystal uses μP, hence Amber uses a shared learning rate while Crystal uses a per-layer learning rate.

Decay to Zero



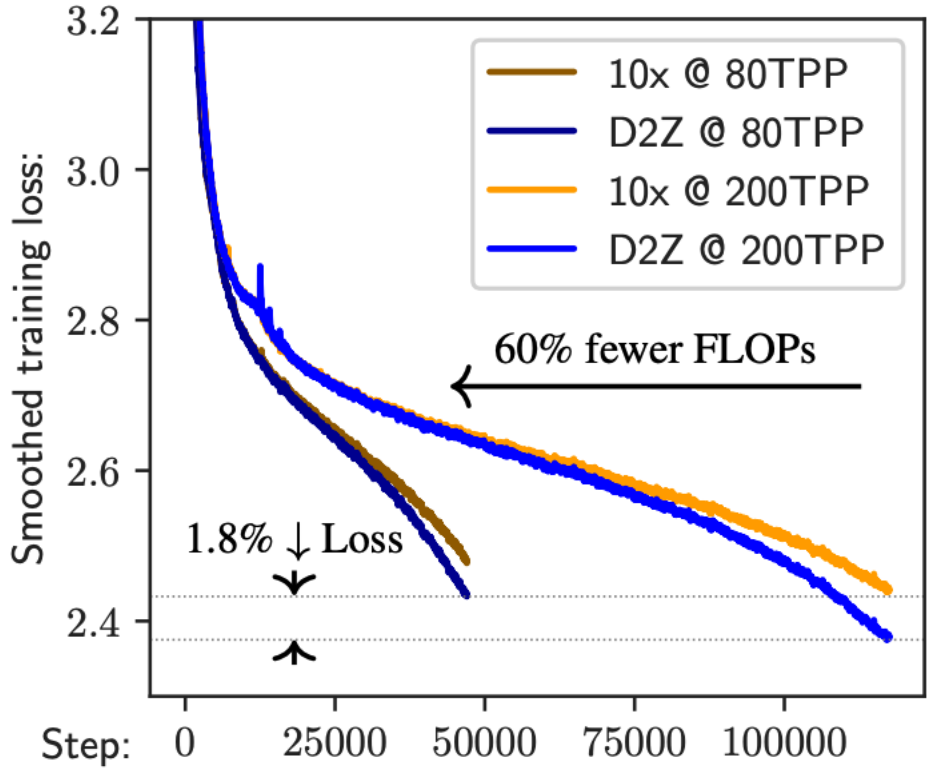


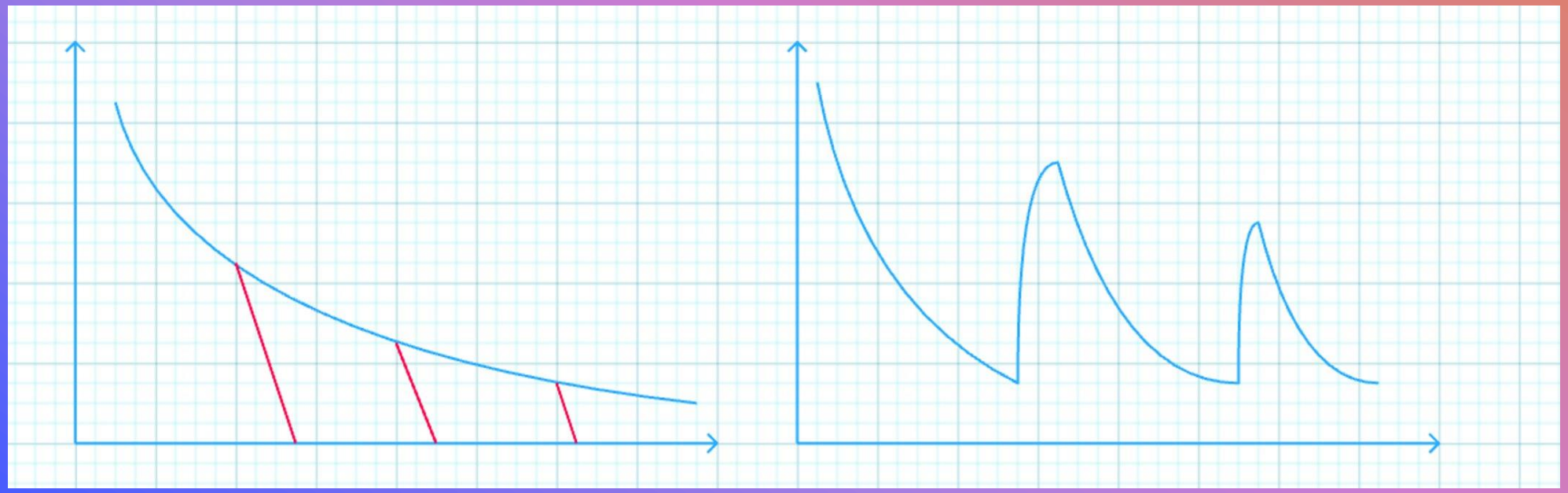
Figure 1: A 610M model trained for 80 TPP with *Linear*-D2Z has better train (and *validation*) loss than when trained for 200 TPP with *Linear*-10×.

[1] STRAIGHT TO ZERO: WHY LINEARLY DECAYING THE LEARNING RATE TO ZERO WORKS BEST FOR LLMS

Ramp-off and Warmups

- In practice, there are other factors affecting learning rate
  - When your datasets come in as multiple stages (may contain distribution shift)
  - When you want to output intermediate checkpoints
  - When you want to do continual training
- Here are some possible strategies:
  - Prepare a learning rate schedule for all the stages, and ramp-off models at each stage
    - Pro: Overall training is smoother, no risk with incorrect warmup
    - Con: Hard to predict the whole learning rate schedule without knowing the full data size
  - Have a learning rate schedule for each stage, and re-warmup at each stage
    - Pro: Easy to plan and implement
    - Con: Warm-up and hyperparameter choices can be tricky

Ramp-off and Warmups



Ramp-off approach, used in Jais-30B model. Prior work such as [1] suggested to linearly decay to zero

Multi-stage warm-up approach, as in LLM360/Crystal. Prior work such as [2] empirically confirms that warmup is necessary in different settings

[1] https://github.com/Stability-AI/StableLM/?tab=readme-ov-file#stablelm-3b-4e1t
[2] Continual Pre-Training of Large Language Models: How to (re)warm your model? arXiv:2308.04014

Precision Curriculum

- Setting the right model precision is another tradeoff*
  - Higher precision means little loss of information and more stable
  - In Transformer, people set the precisions based on the typical ranges of the layers
- Intuitively, models also tend to get more stable as training goes on
  - We empirically find that increasing the precision for latter stages of training do not change the model performance significantly
- We can also view lower precision as a form of quantization
  - Several work in quantization [1,2] suggest that it might be beneficial to have higher precision at early the stage of training

* This tradeoff is also hardware specific, for example, some Nvidia GPUs have special computation units for FP16, FP8. TPUs only support FP32 and BF16.

[1] DSD: Dense-Sparse-Dense Training for Deep Neural Networks, arXiv:1607.04381
[2] Pufferfish: Communication-efficient Models At No Extra Cost, arXiv:2103.03936

Data Weighting and Mix

- Fixed data weighting
  - Methods such as DoReMi [1] target at computing the best data weights.
    - However, we empirically find the DoReMi predicitons are different when using different proxy sizes
  - A more common approach is to find data weighting empirically by performing sweeps
- Dynamic/multi-stage data weighting scheduling
  - Note that most scaling law study assume the dataset is sampled uniformly from the same distribution across training
  - Hence it is possible to estimate data mix with a small proxy model
  - It is uncertain whether we can estimate a shifting data schedule with a proxy
    - Intuitively, larger model may learn certain “ability” with less data, allowing them to be ready for “next step” earlier
    - In LLM360, we attempt to estimate this phenomenon but don’t have enough computation resources to draw conclusions

[1] DoReMi: Optimizing Data Mixtures Speeds Up Language Model Pretraining, arXiv:2305.10429

Data Readiness

- In reality, not everything is ready at the start of training
- LLM training usually spans over weeks if not months
  - New requirement, new data may be available during training
- Since several choices model's training curriculum has been decided in advance, improper planning here will result in suboptimal models
- Practically, we suggest model ramp-off and warm-up strategies as described earlier

Plan the training according to your practical needs

Training Framework

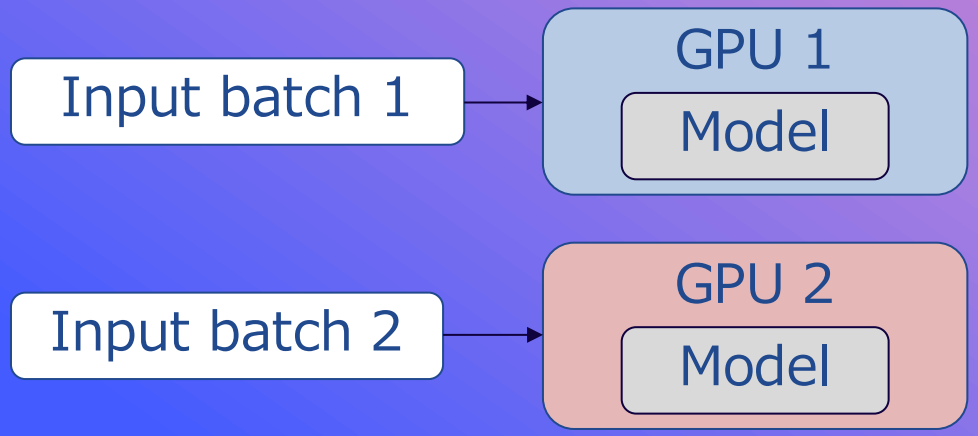
- There are various requirements we need from the training framework:
  - Code stability and implementation correctness
  - Parallelization Support

Background on Parallel Training

1. The input dataset is very large.

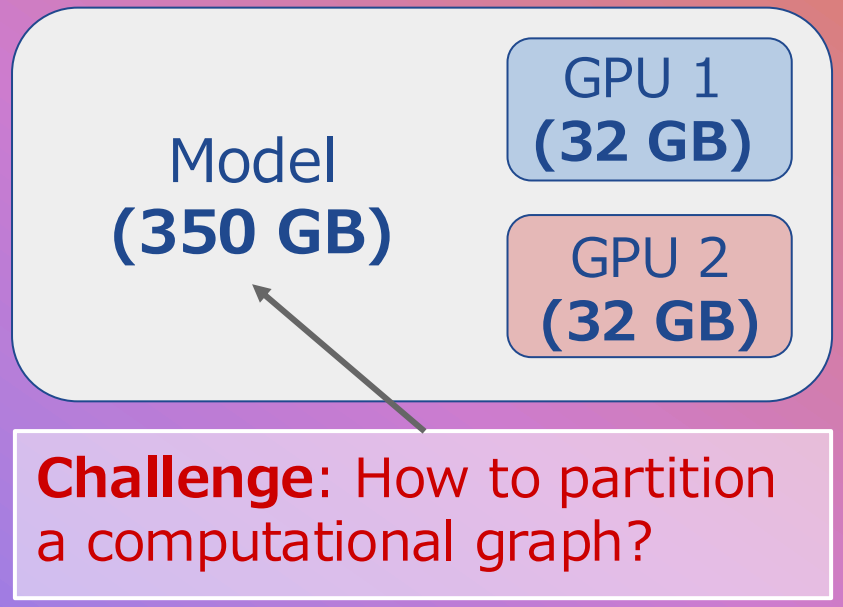
😃 **Easy.**

Data parallelism: partition input data and replicate the model



2. The model is very large.

😖 **Hard !!**



Challenge: How to partition a computational graph?

Heuristic Distributed Params Tuning

An example parallelization tuning for the LLM360 65B model training job.

FSDP v.s. 3D parallelization

- FSDP
  Throughput: 2600*224/s = 582.4k/s
- Megatron MP (tensor-model parallelism); PP (pipeline parallelism):

3D parallelism performance tuning on 56 4xA100 80G DGX nodes (224 GPUs in total).

Config	throughput	Speedups
MP=4; PP=1; mbs=10	4587520/7.81 = 587.39k	~1
MP=2; PP=2; mbs=10	4587520/7.02 = 653.49k	~1.13x
MP=2; MP=2; mbs=2	4587520/7.05 = 650.71k	~1.12x
MP=2; PP=4, mbs=2	4587520/6.92 = 662.94k	~1.15x

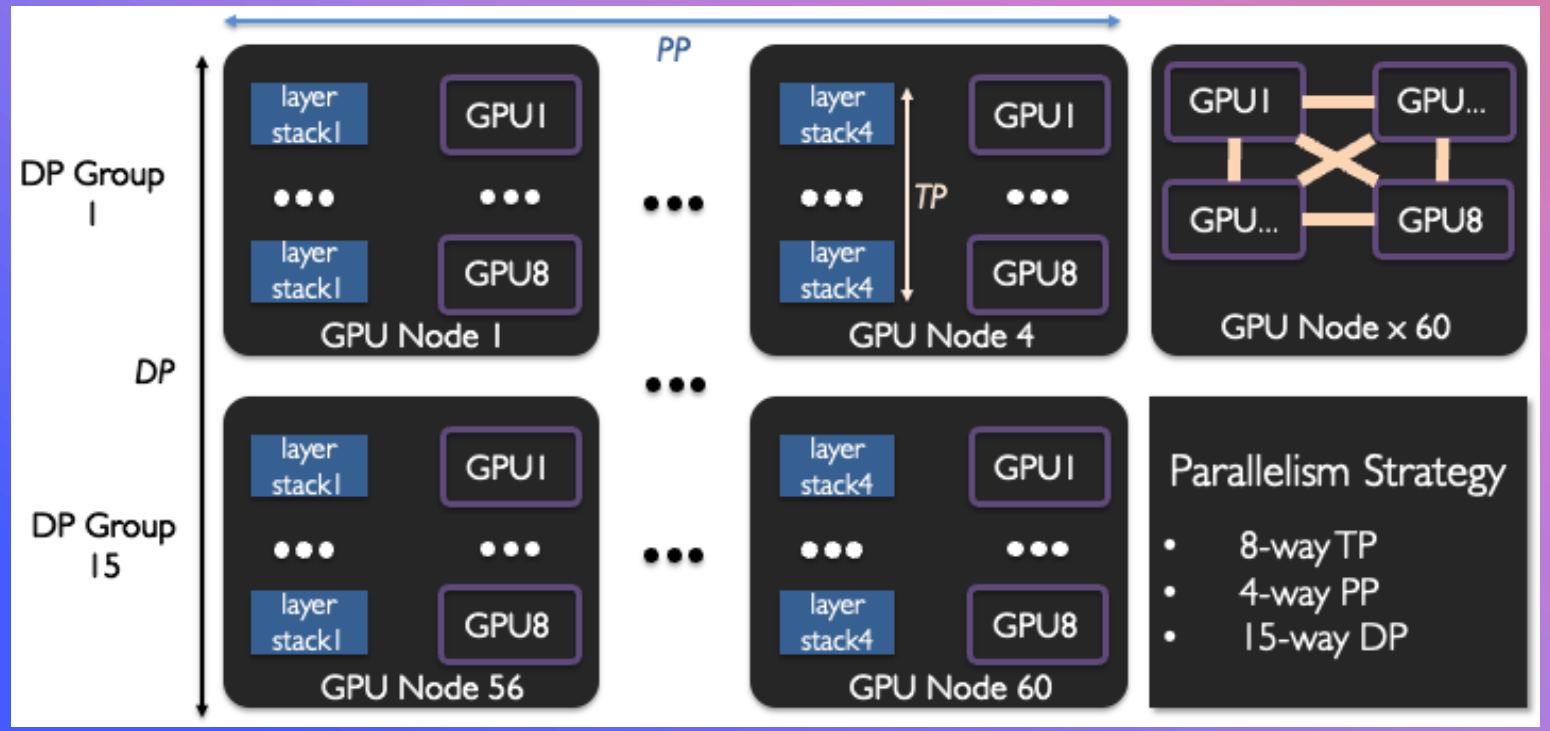
Heuristic Distributed Params Tuning

NUM_NODES	NUM_GPU	TP	PP	DP	micro_bsz	batch runtime	Error	VRAM max	train_time (d)
8	32	1	4	8	1		Out Of Memory		0
8	32	1	4	8	2		Out Of Memory		0
8	32	1	8	4	1	46141		29G	15.27352546
8	32	1	8	4	2		Out Of Memory		0
8	32	1	16	2	1	50994		22G	16.87995833
8	32	1	16	2	2	48636		33G	16.09941667
8	32	1	16	2	4		Out Of Memory		0
8	32	1	32	1	1	62545		19G	20.70355324
8	32	1	32	1	2	59614		30G	19.73333796
8	32	2	2	8	1	58777		35G	19.45627546
8	32	2	2	8	2		Out Of Memory		0
8	32	2	4	4	1	59342		22G	19.64330093
8	32	2	4	4	2	55036		29G	18.21793519
8	32	2	4	4	4	53151		39G	17.59396528
8	32	2	8	2	1	63602		16G	21.05343981
8	32	2	8	2	2	57488		21G	19.02959259
8	32	2	8	2	4	55926		32G	18.51254167
8	32	2	8	2	8		Out Of Memory		0
8	32	2	16	1	1	69863		12G	23.12594676
8	32	2	16	1	2	62218		18G	20.59531019
8	32	2	16	1	4	59400		29G	19.6625
8	32	2	16	1	8		Out Of Memory		0
16	64	1	4	16	1		Out Of Memory		0
16	64	1	8	8	1	24499		29G	8.109622685
16	64	1	8	8	2		Out Of Memory		0
16	64	1	16	4	1		Skip. Can use Larger Batchsize		0
16	64	1	16	4	2	26807		34G	8.873613426
16	64	1	16	4	3		Adjust global bsz to 2064. Out Of Memory		0
16	64	1	16	4	4		Out Of Memory		0
16	64	1	32	2	2		Skip. Can use Smaller PP		0
16	64	1	32	2	4		Out Of Memory		0
16	64	2	2	16	1	31189			10.32413657
16	64	2	2	16	2		Out Of Memory		0
16	64	2	4	8	2		Skip. Can use Larger Batchsize		0
16	64	2	4	8	4	28243		39G	9.348956019
16	64	2	8	4	4		Skip. Can use Smaller PP		0
16	64	2	8	4	8		Out Of Memory		0
16	64	2	16	2	1		Skip. Can use Smaller PP Larger Batchsize		0
16	64	2	16	2	2		Skip. Can use Smaller PP Larger Batchsize		0
16	64	2	16	2	4		Out Of Memory		0
16	64	2	32	1	4		Skip. Can use Smaller PP		0
16	64	2	32	1	8		Out Of Memory		0

- 3D parallelism performance tuning on 8 4xA100 40G DGX nodes (32 GPUs in total).
- It is more important to tune if the cluster is not-so-advanced. We see tuning can achieve 3x different in throughput.

Final Model Parallelism Solution

The training parallelism strategy of K2



DP: Data Parallelism

TP: Tensor Parallelism

PP: Pipeline Parallelism

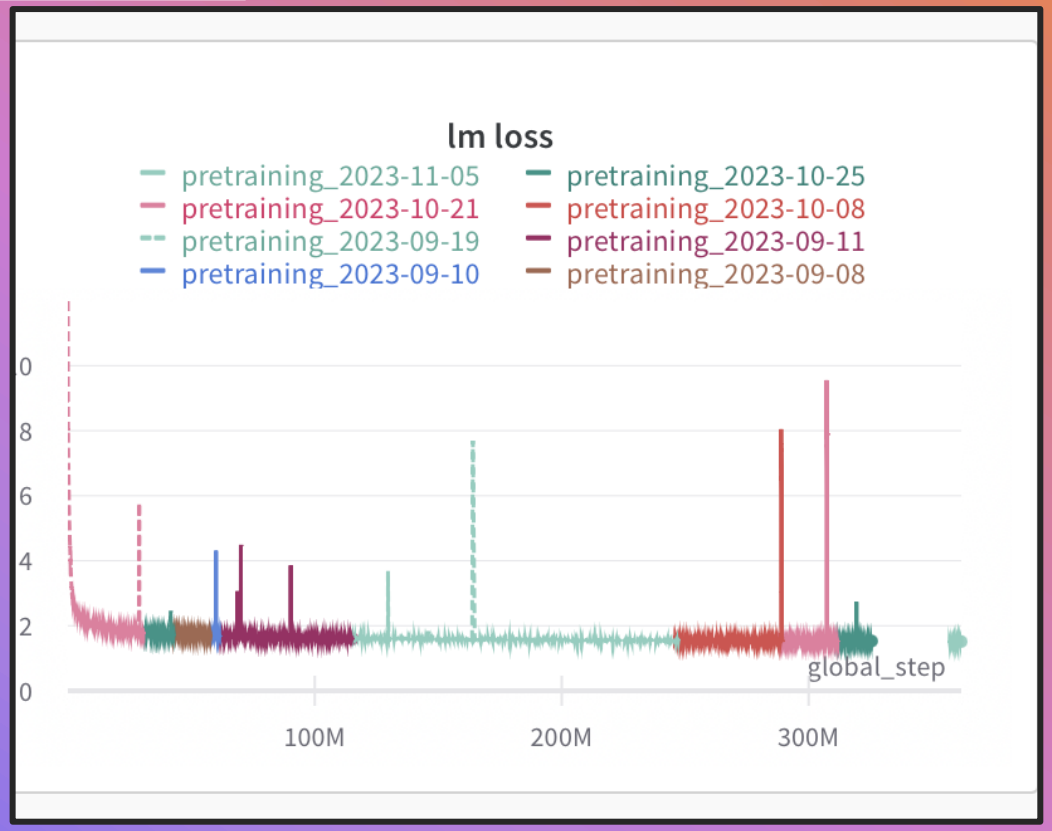
We also used Context Parallelism (CP) along with the TP dimension

LLM360 Developments

- We are currently working on training infrastructures for long context and mixture of expert training.

Fault Tolerance

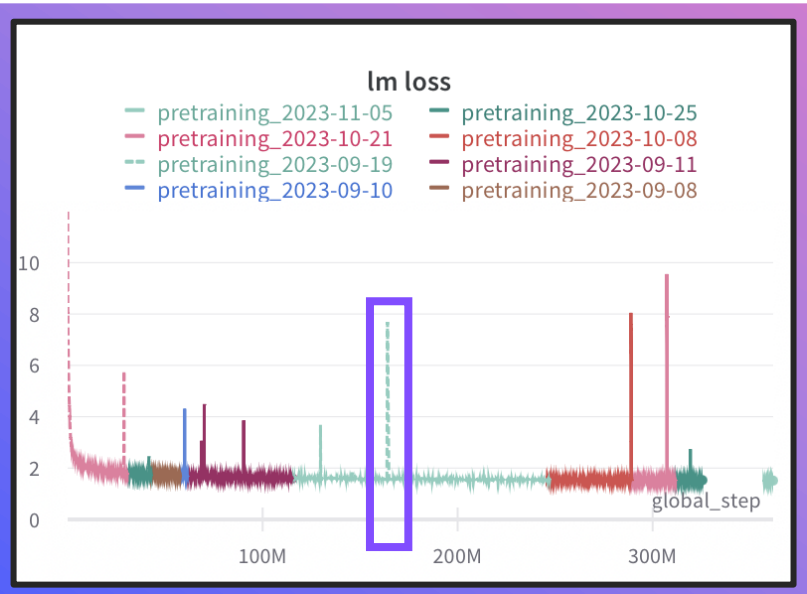
- The LLM pre-training still has many issues:
  - Hardware failure, e.g., CUDA NCCL error.
  - Unknown hardware/network slowing down.
  - Loss spikes during training.
  - NaN loss and training divergence.



Type of Hardware Failure

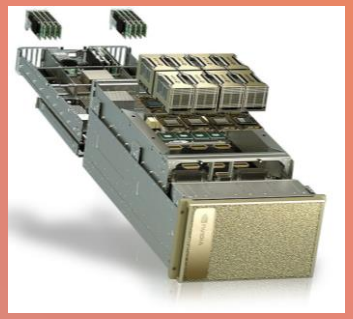
Type of failure	Description
NCCL test timeout	Timeout duration has been exceeded by init container.
Low active tensor core	Low fractions of active tensor core slows training down.
Bad GPU	A GPU is down.
Unhealthy GPU nodes	GPU node-level hardware failure.
OS input/output error	File systems issue/failure.
Lustre error	GPU nodes reboot due to lustre error.
Mount failure	GPU nodes mount failures because he user job is stuck in pending state.
Lack of storage	Running out of disk space on the cluster.

Fault Tolerance



Skip the current data batch when loss spike or NaN loss are observed.

Training GPU Pool



e.g., × 60

Backup GPU Pool

e.g., × 4

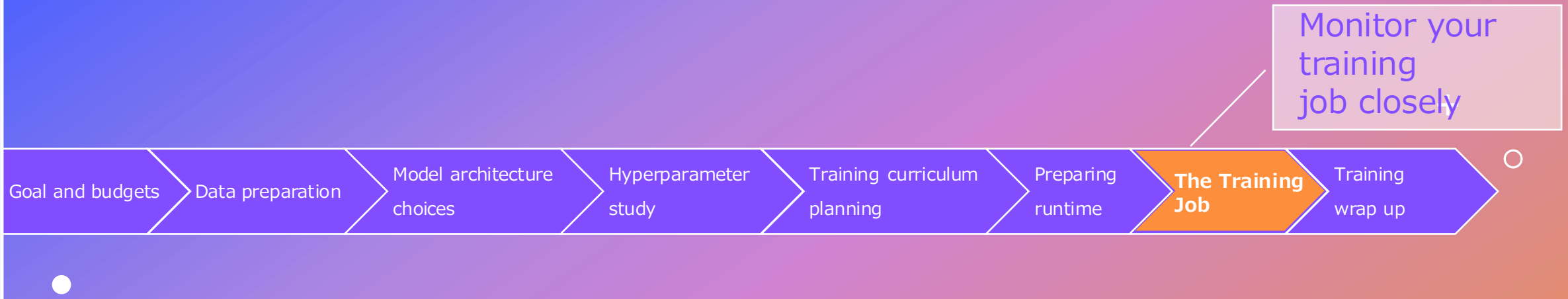
Replacing “failed” node with a new one from the backup GPU pool.

Precision

- Setting precision is a tradeoff between numeric accuracy vs. cost
- By changing the precision (e.g., FP32->FP16 or BF16), the numeric range changes significantly
- Underflow: the smaller exponent range of FP16 causes it easy to underflow
  - Loss scaling can help but not always
  - Loss scaling: multiply the gradient by a large value before back-prop, and scale back before applying the update. (usually used on FP16, FP8)
- BF16 has the same exponent range as FP32, which reduces the risk of underflow
  - BF16 retains more precision for small values, sacrificing precision for large values (like integers larger than 200)
  - We found in the public Megatron-LM implementation, BF16 is used for position index, which causes many nearby positions to have the same value.

Problems Encountered

- During LLM360 training, we have encountered a few problems in runtime
- Configuration bug in Lit-llama repository
  - Models get stored at incorrect precision (FP16) at certain environment
- Incorrect precision in Megatron-LLM repository
  - BF16 is used to store position index, which hits beyond BF16's weak range
- Precision changes midway due to Cerebras hardware upgrade


Monitor your training job closely
Goal and budgets
Data preparation
Model architecture choices
Hyperparameter study
Training curriculum planning
Preparing runtime
The Training Job
Training wrap up

Evaluation and Logging

- Remember to plan enough resources for evaluation
  - One embarrassing lesson we learn during LLM360 is that we don’t have enough evaluation machines for the 65B model
  - Frequent evaluation will ensure we don’t waste time on an ill-behaved model for too long
    - This is again a tradeoff: evaluation resources vs. training resource
- Common evaluation:
  - Held-out perplexity: one of the most direct measure
    - Practically the trend in training loss works well enough (if the data is dedup)
    - Held-out set is still important to ensure no accidental data repetition
  - Benchmark: task-based benchmarks to direct measure desired metrics, just ensure we don’t have data leak
  - Test out generation: trainers should constantly sample output from the model

Benchmarks

Generation

- Example: BigBench, GSM8K, MBPP
- Often involve CoT, or complex generation
- More reliable for generation tasks
- Difficult and high variance for small-scaled models

Multiple Choice

- Example: MMLU, Arc
- Easy to implement and evaluate
- Implemented by:
  - Actual generation: hard to control
  - Perplexity choices: require normalizing over choice length
- Some metrics can be misleading and cannot detect degenerated models

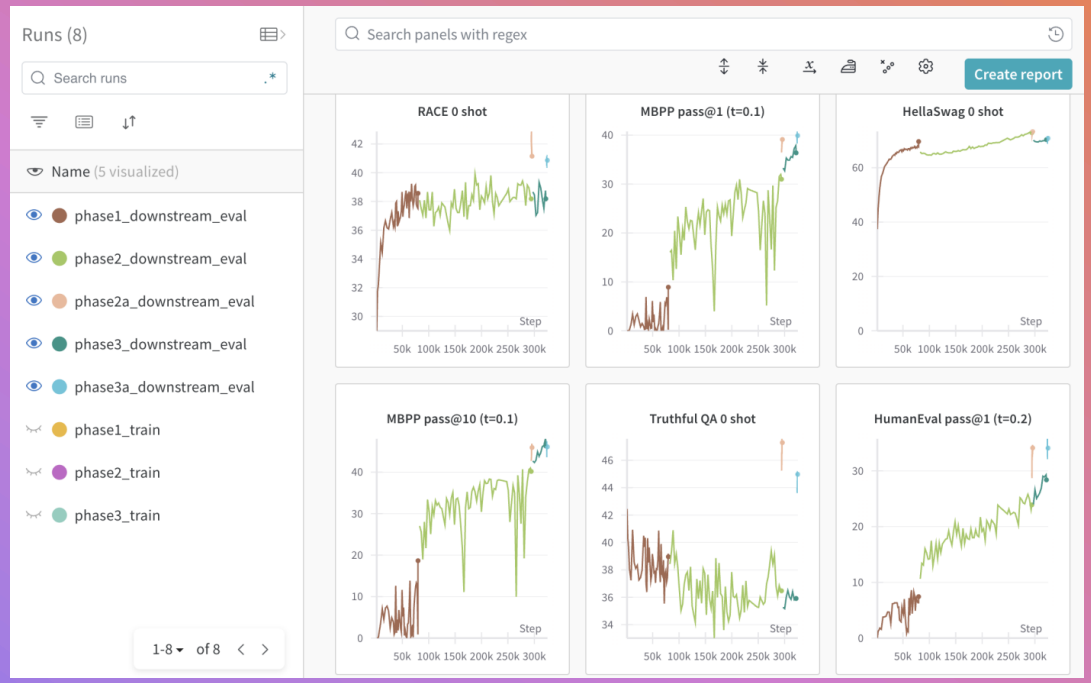
Deal with Problems

- Loss Spikes
  - Loss spikes are common problems during large scale pretraining
    - Empirically, we only observe spikes during our 65B model training but not in the 7B training
  - Solutions to loss spikes
    - Reduce influence of data point, such as embedding layer gradient shrink [1]
    - Simple ones can simply skip a data instance and restart the training
    - Some other methods are also reported to alleviate spikes, such as model averaging [2]

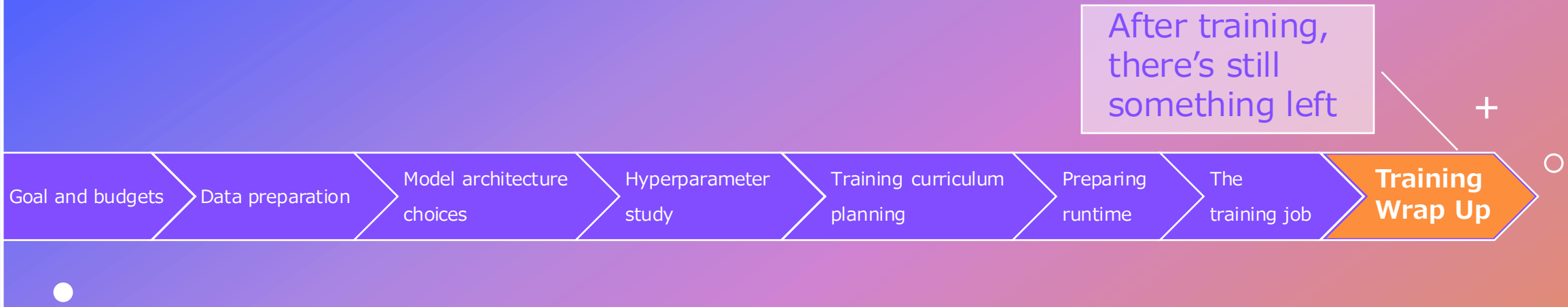
[1] GLM-130B: An Open Bilingual Pre-trained Model arXiv:2210.02414
[2] Early Weight Averaging meets High Learning Rates for LLM Pre-training arXiv:2306.03241

LLM360 Traces

- How do you know if you model behave normally?
  - We provide the LLM360 traces (intermediate checkpoints [1] , model outputs [2] , evaluation results [3])
  - One can compare the training trace with ours as references



[1] https://huggingface.co/LLM360
[2] https://huggingface.co/spaces/LLM360/k2-gallery
[3] https://wandb.ai/llm360


After training, there's still something left
Goal and budgets
Data preparation
Model architecture choices
Hyperparameter study
Training curriculum planning
Preparing runtime
The training job
Training Wrap Up

Quantization

- Quantization allows one to represent the final model in much lower precision (e.g., BP16 to Int4)
  - Some work [1] show that some parts of a model may not be suitable for quantization
- Quantization Aware Training (QAT)
  - QAT models the effects of quantization during training allowing for higher accuracy compared to other quantization methods
  - These techniques are supported by popular learning framework such as PyTorch [2] and Tensorflow [3]

[1] LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale
[2] https://pytorch.org/docs/stable/quantization.html
[3] https://www.tensorflow.org/model_optimization/guide/quantization/training

Finetuning and Alignment

- Finetune an instruction-following or other (i.e., agent) models
  - Finetune for enhance specific abilities: arithmetic, coding
- Tune with safety and culture alignment
  - Larger models tend to overfit on finetuning data a lot more (our natively tuned K2-Model)
- Finetune with vision ability
  - CyrstalVision and K2Vision are coming soon (work led by Dr. Zhiqiang Shen)

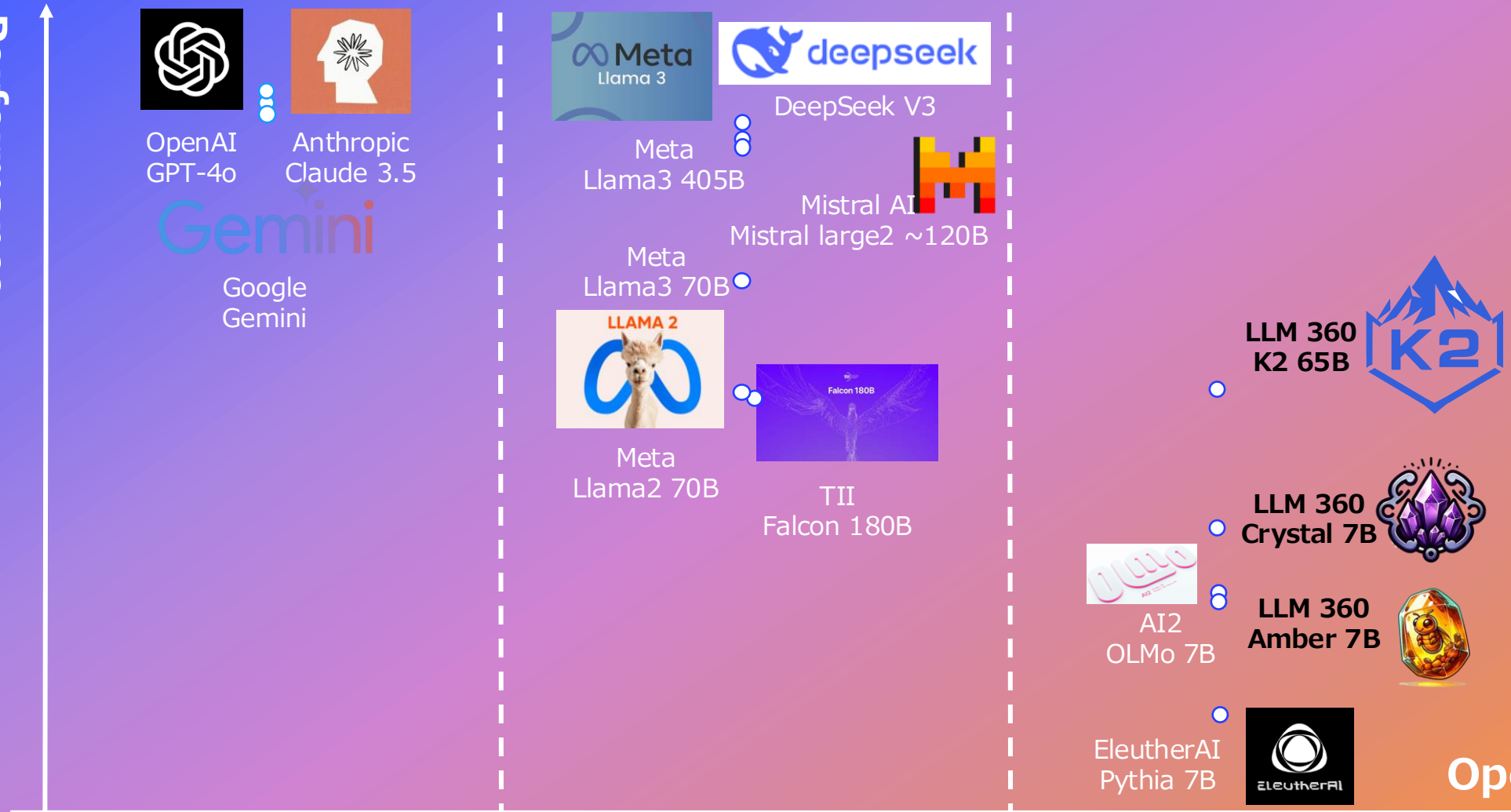
OPEN INNOVATION IN THE FOUNDATION MODEL ERA

What's Open Source?

Open source is source code that is made freely available for possible modification and redistribution. Products include permission to use the source code,[1] design documents,[2] or content of the product. The **open-source model** is a decentralized software development model that encourages open collaboration.[3][4] A main principle of open-source software development is peer production, with products such as source code, blueprints, and documentation freely available to the public. The open-source movement in software began as a response to the limitations of proprietary code. The model is used for projects such as in open-source appropriate technology,[5] and open-source drug discovery.[6][7]

What's the current situation for open sourcing foundation models?

The Landscape of LLMs



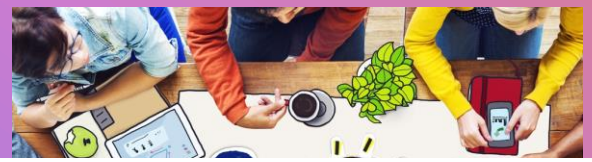
Challenges in Open Science for AI

- Cost of reproduction (especially for LLMs)
- Results sensitive to permutations
- Evaluation is difficult

Even the first step: reproduction and comparison is difficult

Broken Collaboration

- Industry owning most of the training pipeline: the expensive an time-consuming part



Ion Stoica @istoica05 · Feb 11

So given the above, what do you do if you want to win the race of building the best models?

Well, you need a strategy to fully utilize all resources you have to achieve your goal.

1 1 43 7K

Ion Stoica @istoica05 · Feb 11

This means that you need to fully utilize all experts you have, including researchers from academia.

So how can you do this? By having shared artifacts they can all collaborate on. Hence the open source. Hence an infrastructure the academia has access to.

1 3 53 7.2K

- Academy has few access to resources and knowledge about the pipeline

The Open Kitchen Way

Transparent:

Not just serve food (weight), but show the process

Reproducible:

With the recipe and intermediate steps, you can reproduce any step, with the infra (cookware) provided, no secret sauce (training on test sets)

Accessible:

The artifacts need to be accessible, instead of “open” but behind a secret paywall

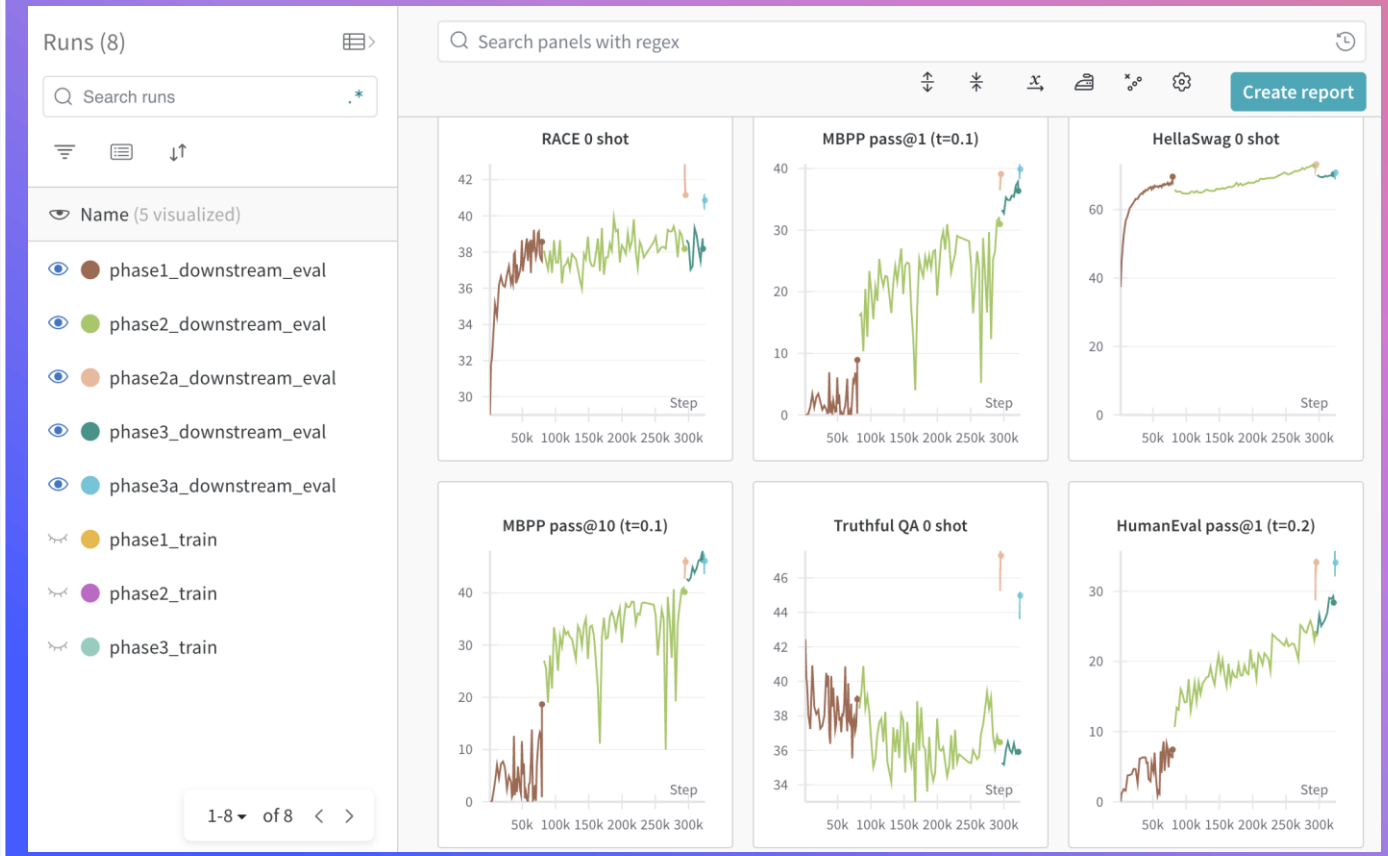
LLM360: Towards Open Source AI

- Even the playground via knowledge sharing
  - Provide artifacts for reproduction and collaboration
  - Enable research directions
  - Reduce repeated work/reduce carbon footprint
- Build tools and standards to enable adoption

Artifacts released for LLM360 models

- Training Code: https://github.com/llm360/
- (up to) 360 Checkpoints: https://huggingface.co/LLM360
- Exact Data Sequence
- Evaluation Trace: https://wandb.ai/llm360/
- And more... https://www.llm360.ai

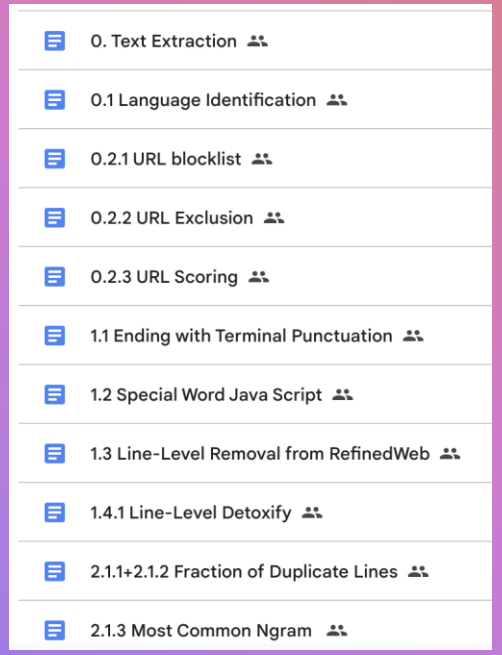
The LLM360 Project

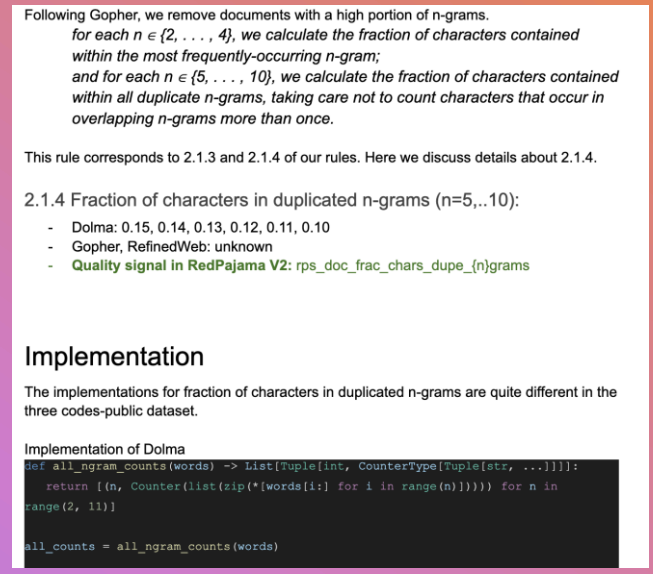


- By releasing the full trace of each model, one can traverse the training timeline and zoom into any step:
  - Check model behavior
  - Conduct ablation study
  - Refer to model statistics

LLM360: Community Ablation

- In this talk, we will see many decisions require supports of empirical decisions
  - E.g., data cleaning strategy, data weighting,
  - Full ablation of these decisions are cost prohibited
- We provide data points and decisions as references
  - Future work can ablate on some of the decisions



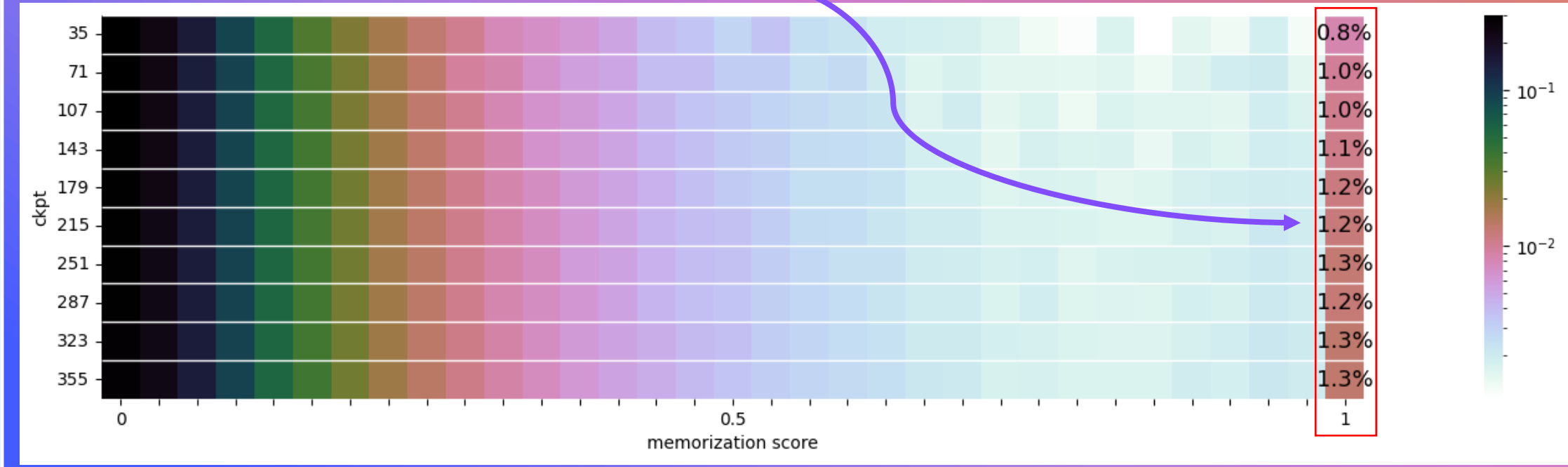


List of Data Processing Decisions for TxT360 Dataset. Ablation study on all of them is not possible for a single team.

Case Study: the Pythia Memorization Study

The figure shows the distribution of memorization scores for 10 selected checkpoints, and annotate the percentage of score = 1 (indicating the sequence is memorized*):

1. More than 1of the sequences 10 are memorized from AMBER
2. AMBER can memorize more sequences with the training going
3. Since we consider more than 32 tokens memorized as equally memorized, the spike at score = 1, indicating that AMBER can memorize a much larger number of tokens than 32

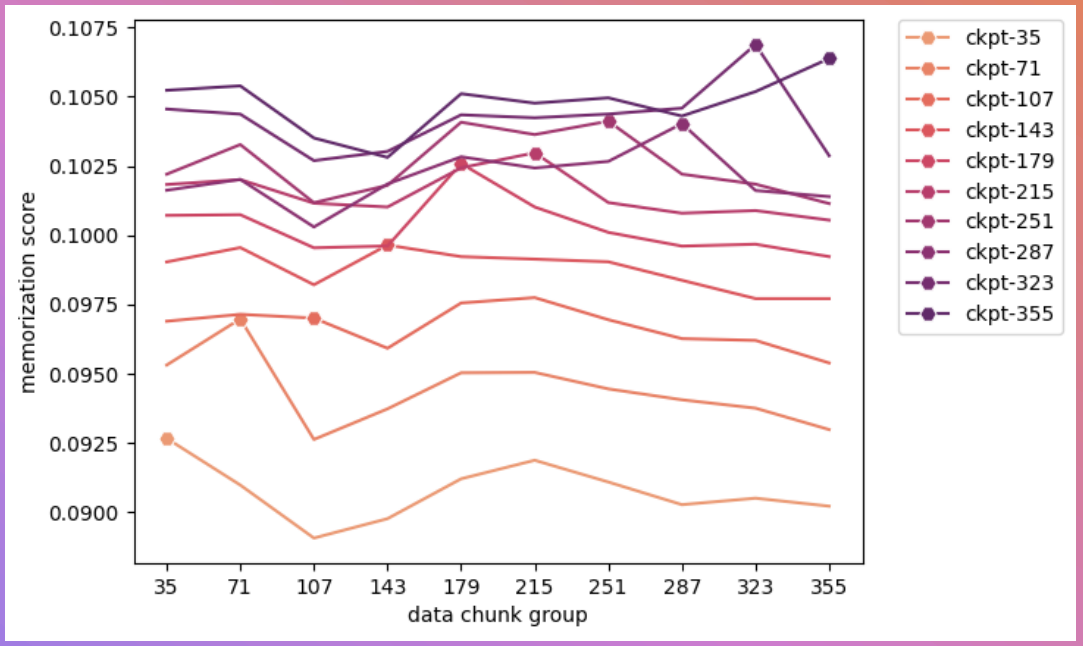


Heatmap indicating proportion of sequence memorized

** Memorization is defined with k-extractible (k=32). A string s is said to be k-extractible if it exists in the training data; or is generated by the language model by prompting with k prior tokens.*

Case Study: the Pythia Memorization Study

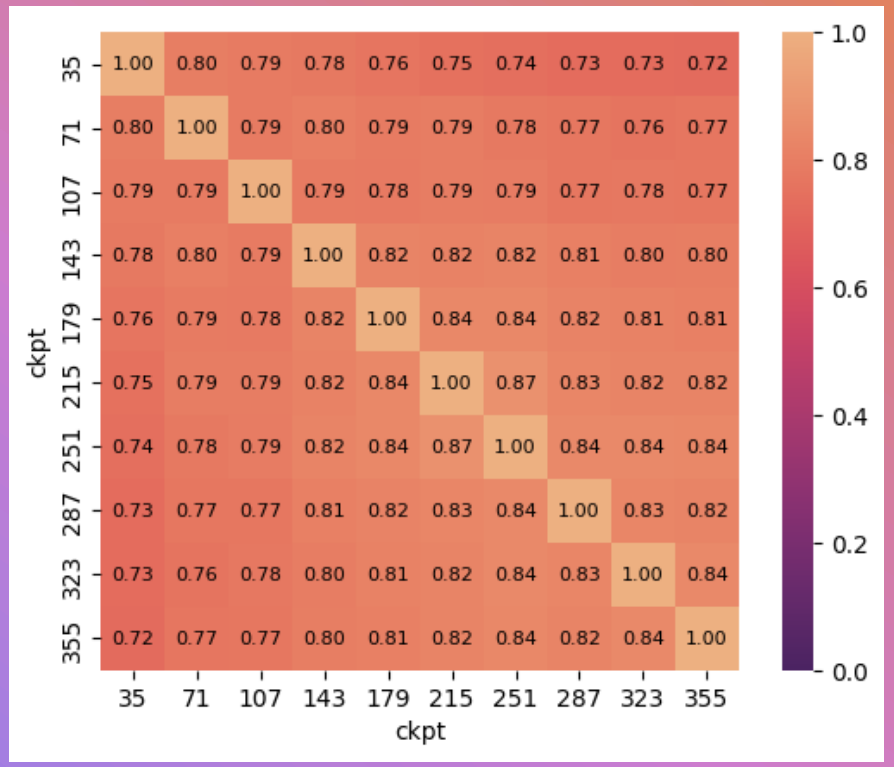
- We group the data chunks according to the selected checkpoints, and plot the memorization score on each data chunk group for each checkpoint
  - AMBER checkpoints memorize the latest seen data much more than previous data
  - For each data chunk, the memorization score drops a bit with additional training, but keeps increasing afterwards.



Memorization score on data chunk for each checkpoint. The marked spots indicate the latest chunk seen by that checkpoint. The part on right of each mark indicates unseen data.

Case Study: the Pythia Memorization Study

- We show the correlation between sequences in terms k-extractible
  - Note that all sequences are seen by all the checkpoints, even the least trained one
  - The correlation across checkpoints is strong



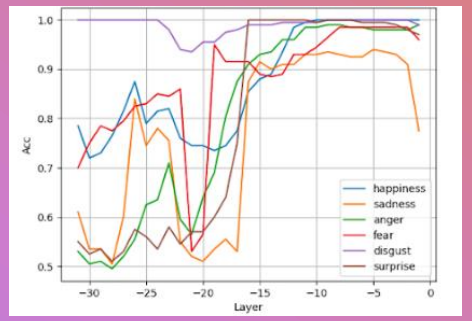
Heat maps visualizing the correlation between which sequences are memorized by different checkpoints.

In the original memorization paper: the authors wrote “hope that future work may replicate some of our findings on entirely distinct corpora”. In LLM360, we confirm these findings.

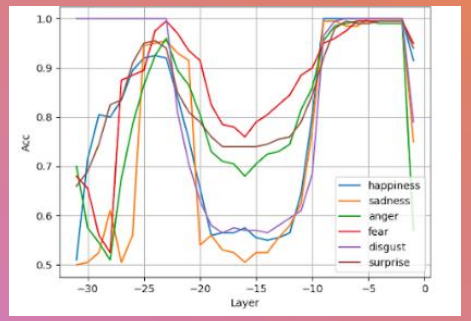
Case Study: Representation over time

- Analyze the representation over time experiment
  - The Representation Engineering [1] method allow one to find relevant neurons to cognitive phenomena in neural networks
  - We perform the RepE scan method on LLM360/Crystal models

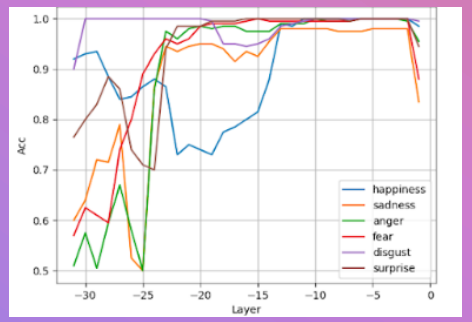
4%



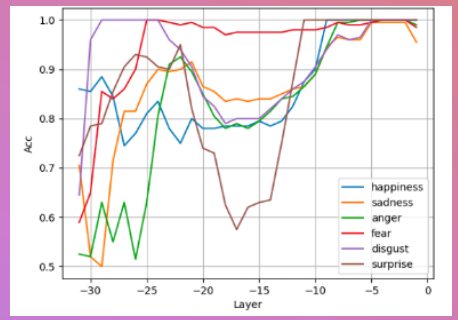
25%



50%



85%



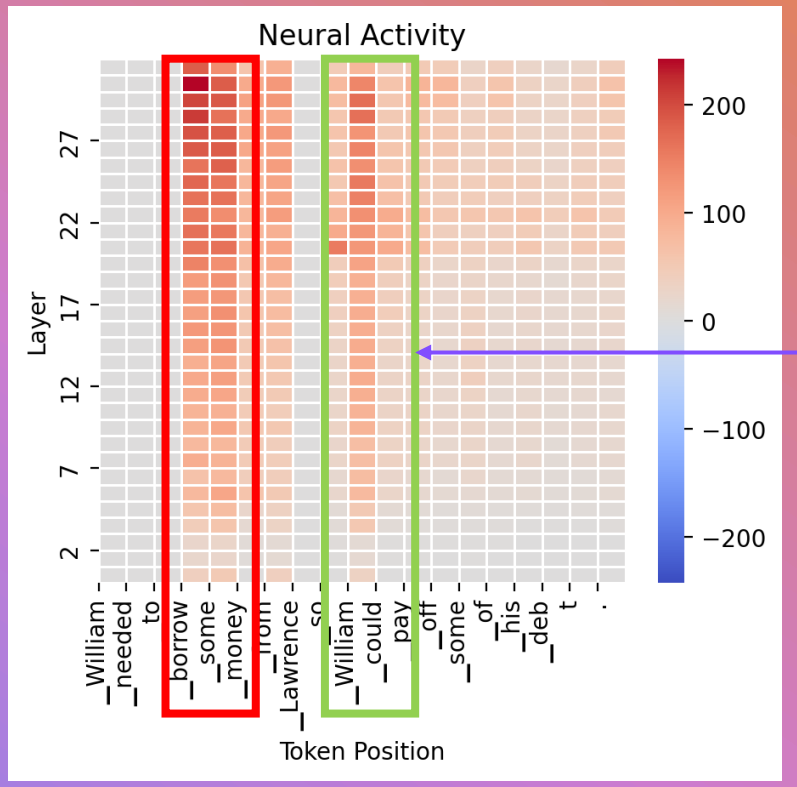
Some initial results: we plot the accuracy using different layer's hidden neurons at predicting different emotions, larger value are closer to the output layer.

1. The reading vectors from higher layers and some lower layers predicts the emotion better
2. With the training going on, more layers start to obtain the cognitive ability

[1] Representation Engineering: A Top-Down Approach to AI Transparency. arXiv:2310.01405

Case Study: Winograd Schema

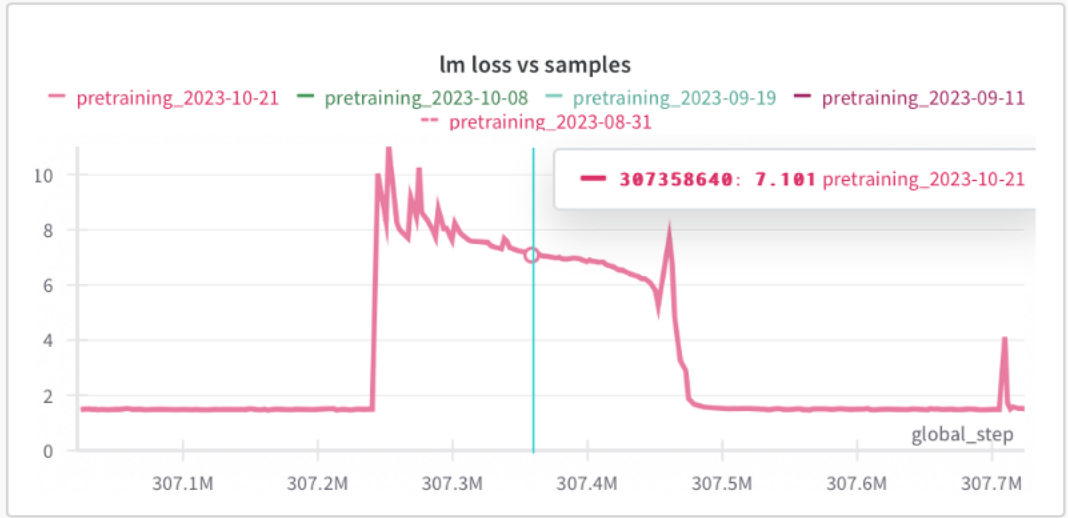
- The Winograd Schema asks the model to resolve an anaphora task with common sense reasoning
  - William needed to **borrow** some money from Lawrence so William could pay off some of his debt.
  - William needed to **lend** money to Lawrence so Lawrence could pay off some of his debt.
- Winograd schema are often in pairs, and one small change can cause significant change in semantics
  - This allows one to study the small but significant change
  - We can further utilize the pair to localize the differences of the internal activations.

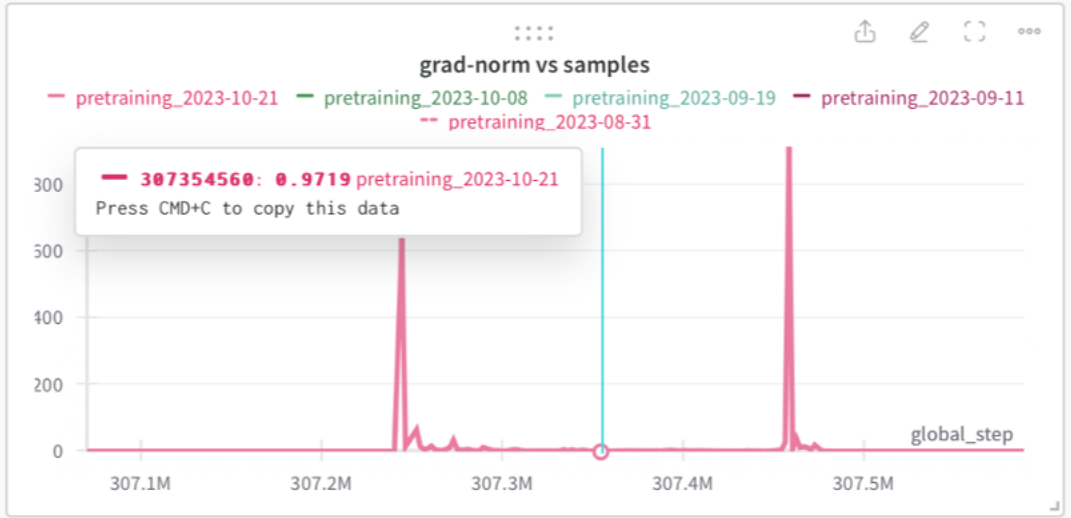


- Utilizing the pair structure, we take the activation value from both sentences and take the differences.
- This cancels out most of the same activities and sheds light on how an LLM solves the Winograd Schema

Optimizer Study for learning behaviors

- Researchers are using K2 spike checkpoints and optimization states to study the learning behaviors





- https://wandb.ai/llm360/k2

LLM360 Projects

Amber: 7B English Model

- The first model of the LLM360 project.

Crystal: 7B English Model that also excels at Code

- More token efficient than the Llama series
- A better balance between coding and language

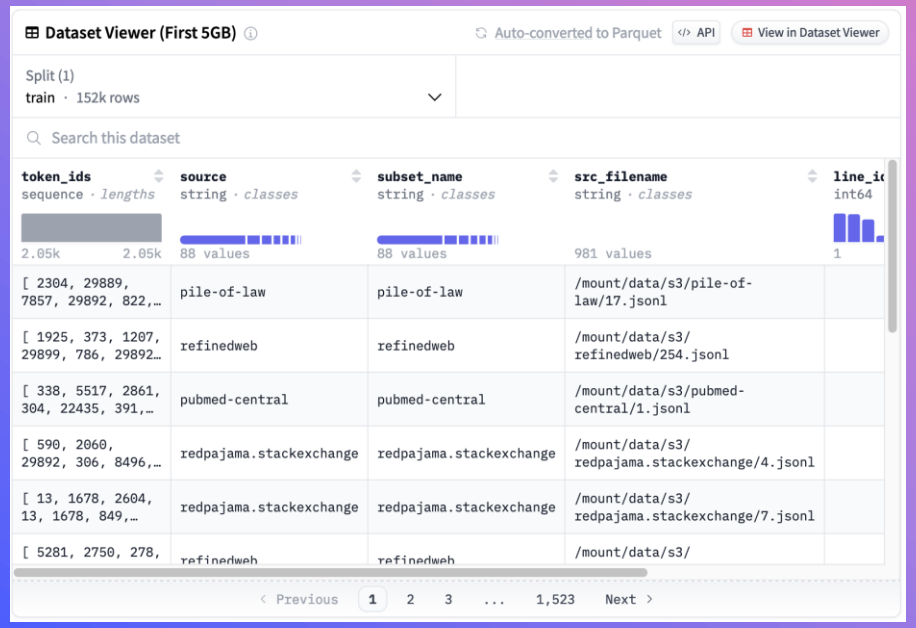
Reproducible large language model at Llama 2 70B level, with 35% less compute

The first dataset to globally deduplicate 99 CommonCrawl snapshots and 14 high-quality data sources, enables precise control over data distribution

K2 Research Artifacts: Training Data

Training data is a closely held secret by enterprises such as OpenAI and Meta.

K2 openly shares all data to advance understanding into data mixtures to optimally and sustainably train and deliver the next generation of LLMs.



Released K2 Artifacts

1.4T **fully open data sequence artifacts** for advanced understanding into data mixtures and to kickstart optimal and sustainable future training

+ 120 **intermediate checkpoints** are made available to empower research into training dynamics

+ 100+ **prompts and output** showing how the model responses change over training lifetime: huggingface.co/spaces/LLM360/k2-gallery

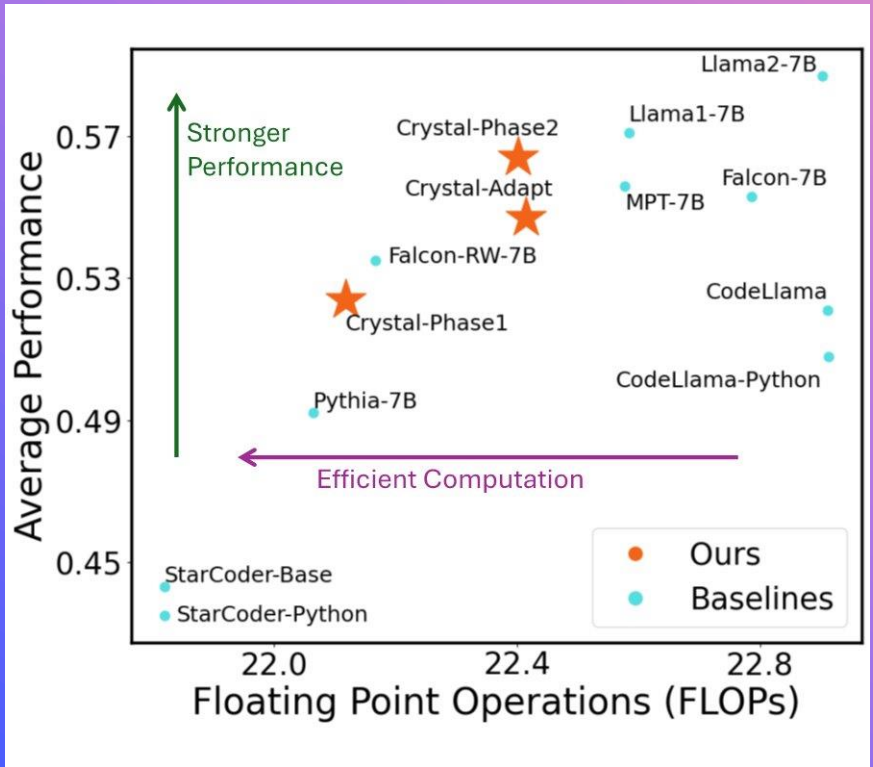
+ 40+ **metric curves** collected through out training lifetime and made publicly available on Weights&Biases: wandb.ai/llm360/k2

+ 21 **evaluation metrics** showing model holistic performance output over training lifetime: huggingface.co/spaces/LLM360/k2-eval-gallery

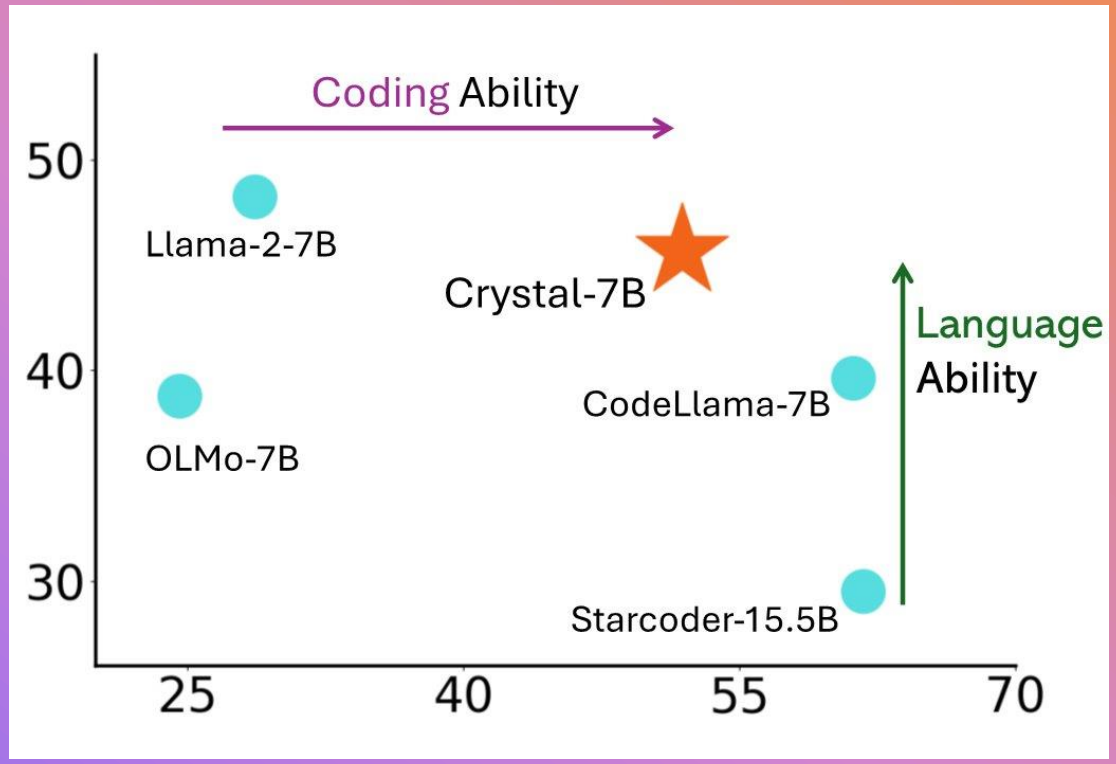
---

= 16TB+ **collection size of model artifacts**, the most complete set of ever released

Overview of Model: Crystal



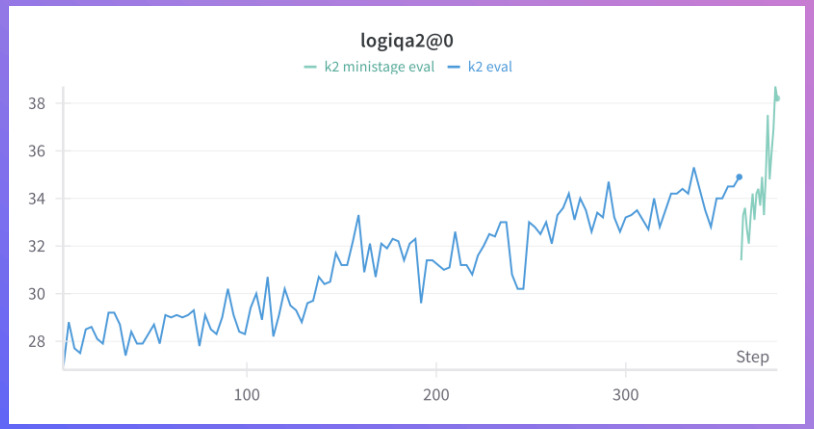
LLM performance through the lens of FLOPs. Crystal achieves a better token efficiency than many of the baselines.

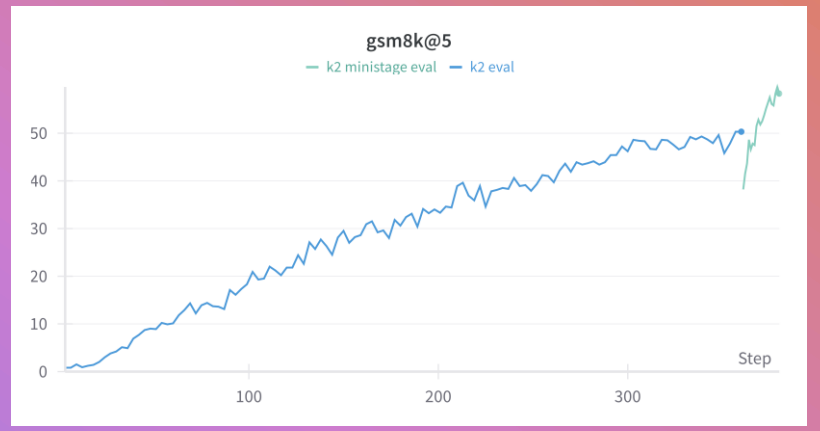


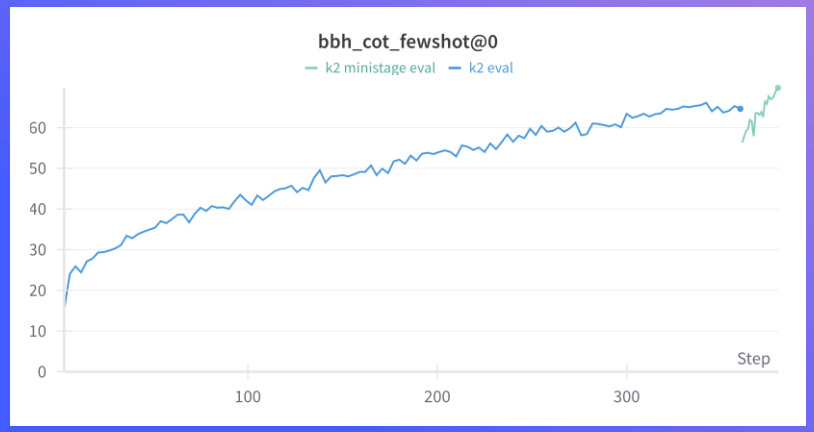
Crystal also achieves a better balance between language and coding ability

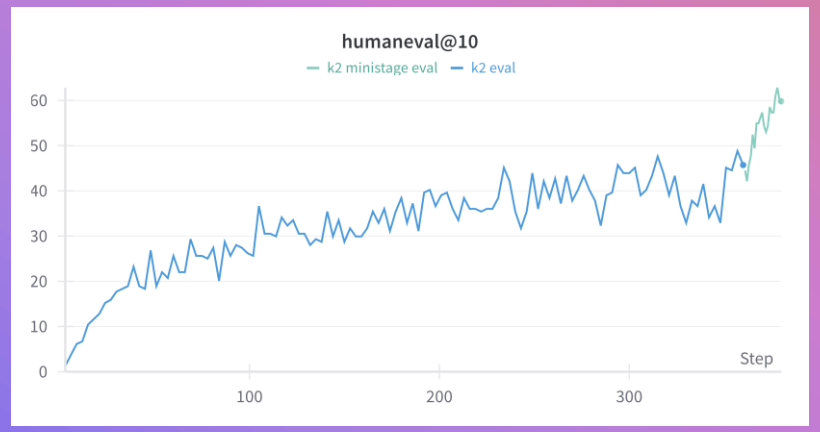
K2 Results

Logic, reasoning, math, and coding ability steadily improved throughout training

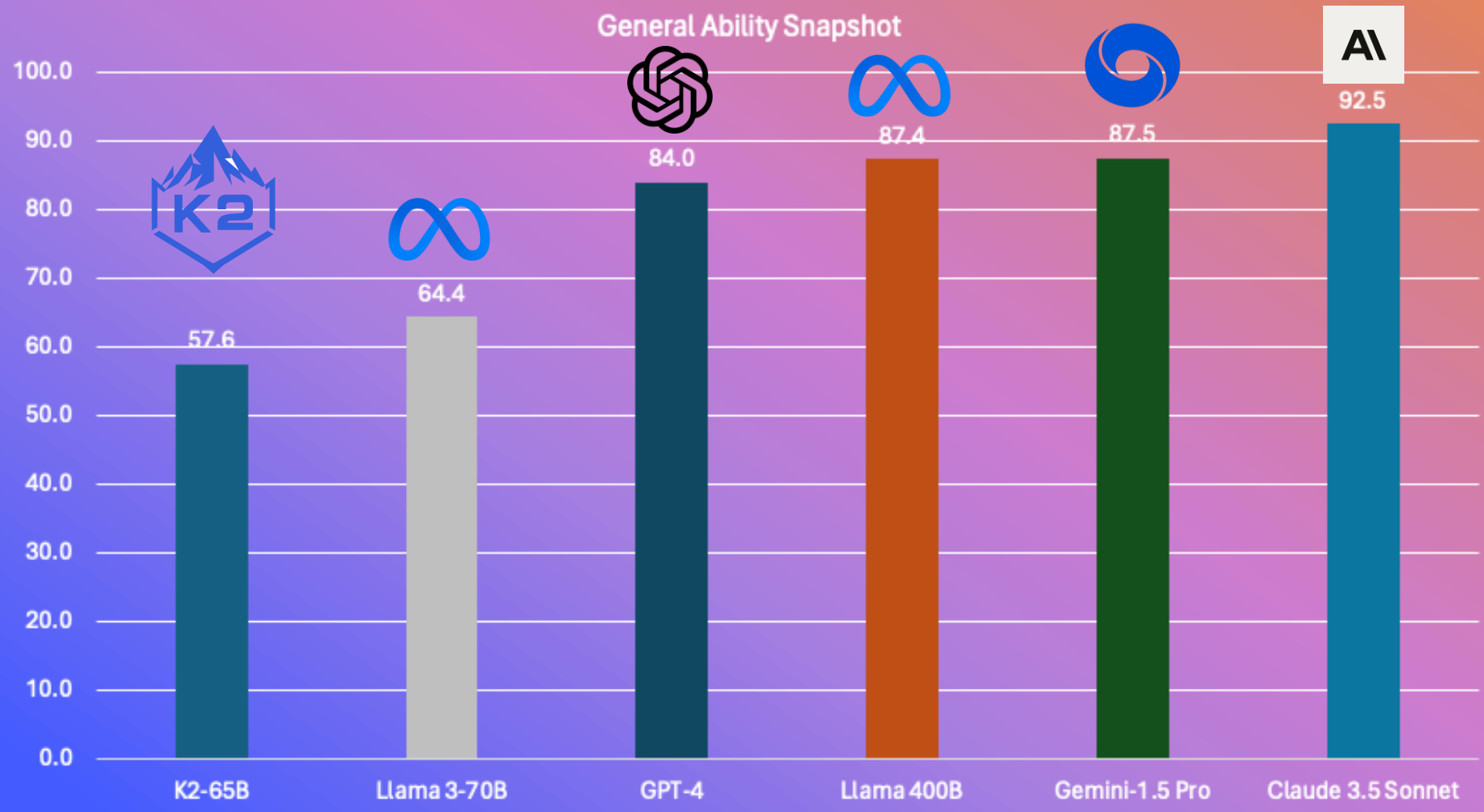






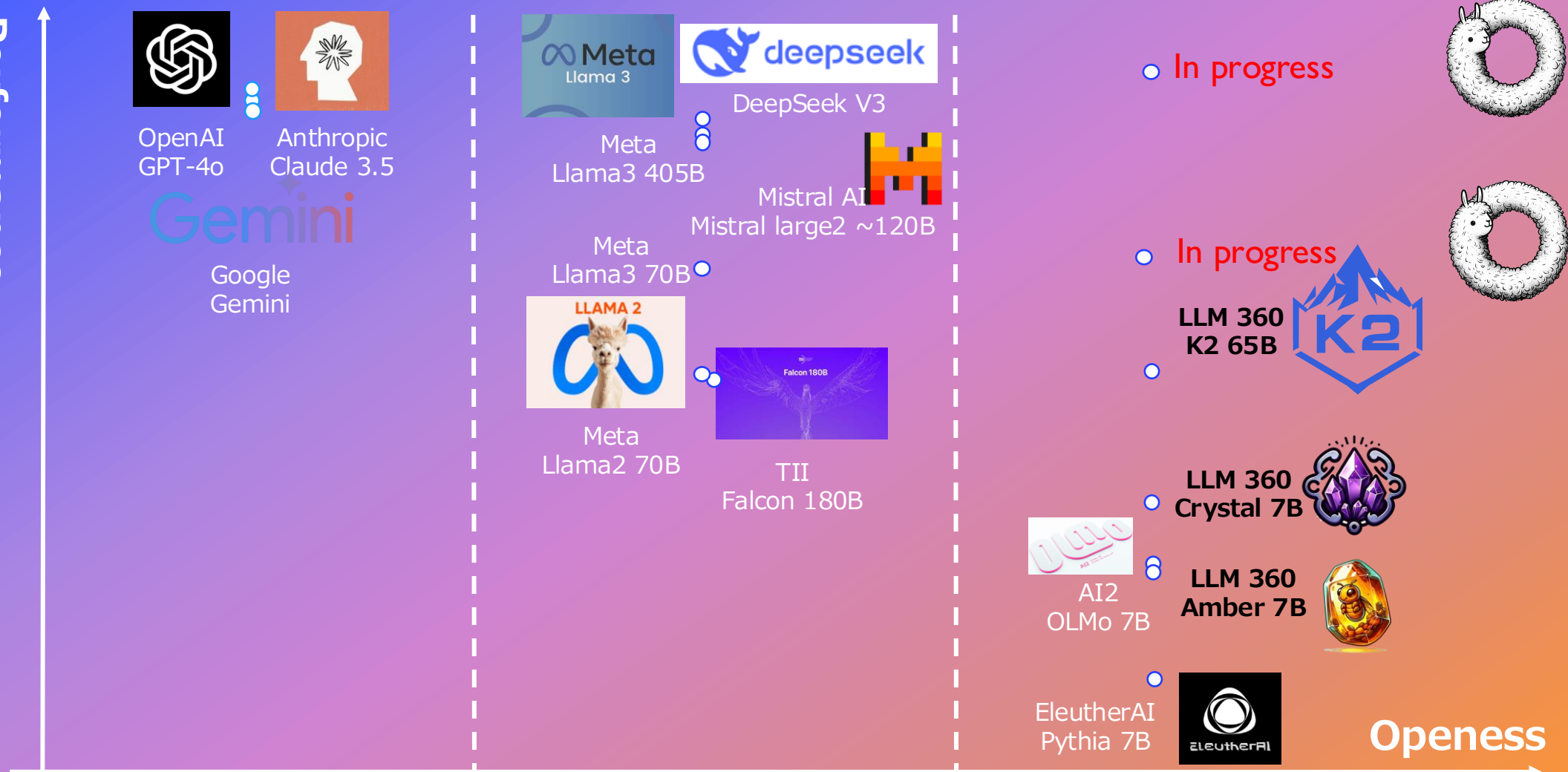


Closing the Gap to SOTA



Average of GSM8K, BBH, HumanEval, and MMLU evaluations

The Landscape of LLMs



THANK YOU