CS 839: Foundation Models
Efficient Training

Fred Sala

University of Wisconsin-Madison

Oct. 9, 2025

Announcements

- **Logistics:**
  - Homework 2 in progress.
  - **Sign up for presentations!**
  - Project information coming out shortly.
- Class roadmap:

Thursday Oct. 9	Efficient Training
Tuesday Oct. 14	Efficient Inference
Thursday Oct. 16	Evaluation
Tuesday Oct. 21	Agents
Tuesday Oct. 23	More Reasoning

Outline

- **Finish Up Last Time**
  - RL training, RLVR, GRPO, reasoning tasks
- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Start Efficient Inference**
  - Speculative decoding, early-exit strategies, Flash decoding

Outline

- **Finish Up Last Time**
  - RL training, RLVR, GRPO, datasets
- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Start Efficient Inference**
  - Speculative decoding, early-exit strategies, Flash decoding

RL Outside of Alignment

- Let’s get back to building **a good model**-doesn’t need to be within the context of alignment
  - This means we don’t have human preference data, but potentially something else

Where does RL fit in here?

- And what are the new reward models going to look like?
- One simple approach: “rewards” for just the correct answers
  - But, unlike in the supervised case, not just one solution

Back to RL: PPO Details

- Note that we could directly apply PPO to train
- We would integrate some notion of verifier correctness into the reward
- Let's dive a bit deeper into PPO

$$\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

- Two forms

(that we can combine)

Advantage $A_t = Q(s_t, a_t) - V(s_t)$

$$\hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

PPO to GRPO

- GRPO (Group Relative Policy Optimization)
  - Shao et al, DeepSeekMath

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right),$$

- Most elements are the same compared to PPO, but note that we sample a **group** of G responses.
- Advantage:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \cdots, r_G\})}{\text{std}(\{r_1, r_2, \cdots, r_G\})}$$

GRPO/DeepSeek R1 Rewards

- How to use verifiers in rewards?

$$A_i = \frac{r_i - mean(\{r_1, r_2, \cdots, r_G\})}{std(\{r_1, r_2, \cdots, r_G\})}$$

Very simple: DeepSeek R1 uses:

- **Accuracy rewards**: The accuracy reward model evaluates whether the response is correct. For example, in the case of math problems with deterministic results, the model is required to provide the final answer in a specified format (e.g., within a box), enabling reliable rule-based verification of correctness. Similarly, for LeetCode problems, a compiler can be used to generate feedback based on predefined test cases.
- **Format rewards**: In addition to the accuracy reward model, we employ a format reward model that enforces the model to put its thinking process between '<think>' and '</think>' tags.

Note the thinking tokens!

Strong Performance on Math

AIME Results:

Overall | **AIME 2025 I** | AIME 2025 II | HMMT February 2025 | USAMO 2025

Model	Acc	Cost	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
gemini-2.5-pro ⚠	83.33%	N/A															
o3-mini (high)	80.00%	$3.19															
o1 (medium)	78.33%	$44.40															
o3-mini (medium)	73.33%	$1.67															
DeepSeek-R1	65.00%	$4.91															
QwQ-32B ⚠	60.00%	$1.24															
DeepSeek-V3-03-24 ⚠	53.33%	$0.25															
o3-mini (low)	53.33%	$0.62															
DeepSeek-R1-Distill-32B	53.33%	N/A															
gemini-2.0-flash-thinking	51.67%	N/A															
DeepSeek-R1-Distill-14B	50.00%	$1.15															
DeepSeek-R1-Distill-70B	50.00%	$1.35															
Claude-3.7-Sonnet (Think) ⚠	46.67%	$22.17															
QwQ-32B-Preview	36.67%	$0.58															
gemini-2.0-flash	30.00%	$0.06															

https://matharena.ai/

And Physics

Theoretical Physics Benchmark (TPBench) - a Dataset and Study of AI Reasoning Capabilities in Theoretical Physics

Daniel J.H. Chung[1], Zhiqi Gao[2], Yurii Kvasiuk[1], Tianyi Li[1], Moritz Münchmeyer[1,5], Maja Rudolph[3], Frederic Sala[2], and Sai Chaitanya Tadepalli[4]

[1]Department of Physics, University of Wisconsin-Madison
[2]Department of Computer Science, University of Wisconsin-Madison
[3]Data Science Institute (DSI), University of Wisconsin-Madison
[4]Department of Physics, Indiana University, Bloomington
[5]NSF-Simons AI Institute for the Sky (SkAI), Chicago

February 25, 2025

Abstract

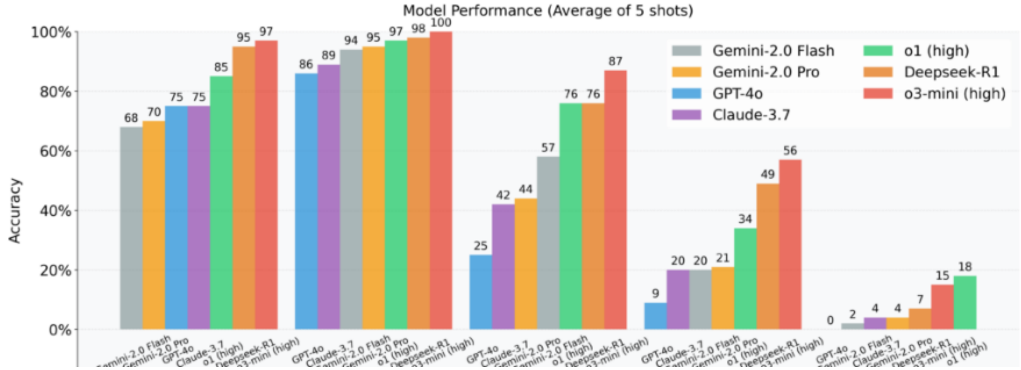
We introduce a benchmark to evaluate the capability of AI to solve problems in theoretical physics, focusing on high-energy theory and cosmology. The first iteration of our benchmark consists of 57 problems of varying difficulty, from undergraduate to research level. These problems are novel in the sense that they do not come from public problem collections. We evaluate our data set on various open and closed language models, including o3-mini, o1, DeepSeek-R1, GPT-4o and versions of Llama and Qwen. While we find impressive progress in model performance with the most recent models, our research-level difficulty problems are mostly unsolved. We address challenges of auto-verifiability and grading, and discuss common failure modes. While currently state-of-the art models are still of limited use for researchers, our results show that AI assisted theoretical physics research may become possible in the near future. We discuss the main obstacles towards this goal and possible strategies to overcome them. The public problems and solutions, results for various models, and updates to the data set and score distribution, are available on the website of the dataset tpbench.org.

TP Bench – Theoretical Physics Benchmark for AI

TPBench is a curated dataset and evaluation suite designed to measure the reasoning capabilities of AI models in theoretical physics. Our test problems span multiple difficulty levels—from undergraduate to frontier research—and cover topics such as cosmology, high-energy theory, general relativity, and more. By providing a unified framework for problem-solving and auto-verifiable answers, TPBench aims to drive progress in AI-based research assistance for theoretical physics.

Read the TPBench Paper on arxiv
Access Public Dataset on Huggingface

Current Model Performance



Break & Questions

Outline

- **Finish Up Last Time**
  - RL training, RLVR, GRPO, datasets
- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Start Efficient Inference**
  - Speculative decoding, early-exit strategies, Flash decoding

Training Foundation Models: **Scale**

Llama family of models,

- *"we estimate that we used 2048 A100-80GB for a period of approximately 5 months to develop our models"*

OPT (Open Pre-trained Transformers),

- *"training OPT-175B on 992 80GB A100 GPUs"*

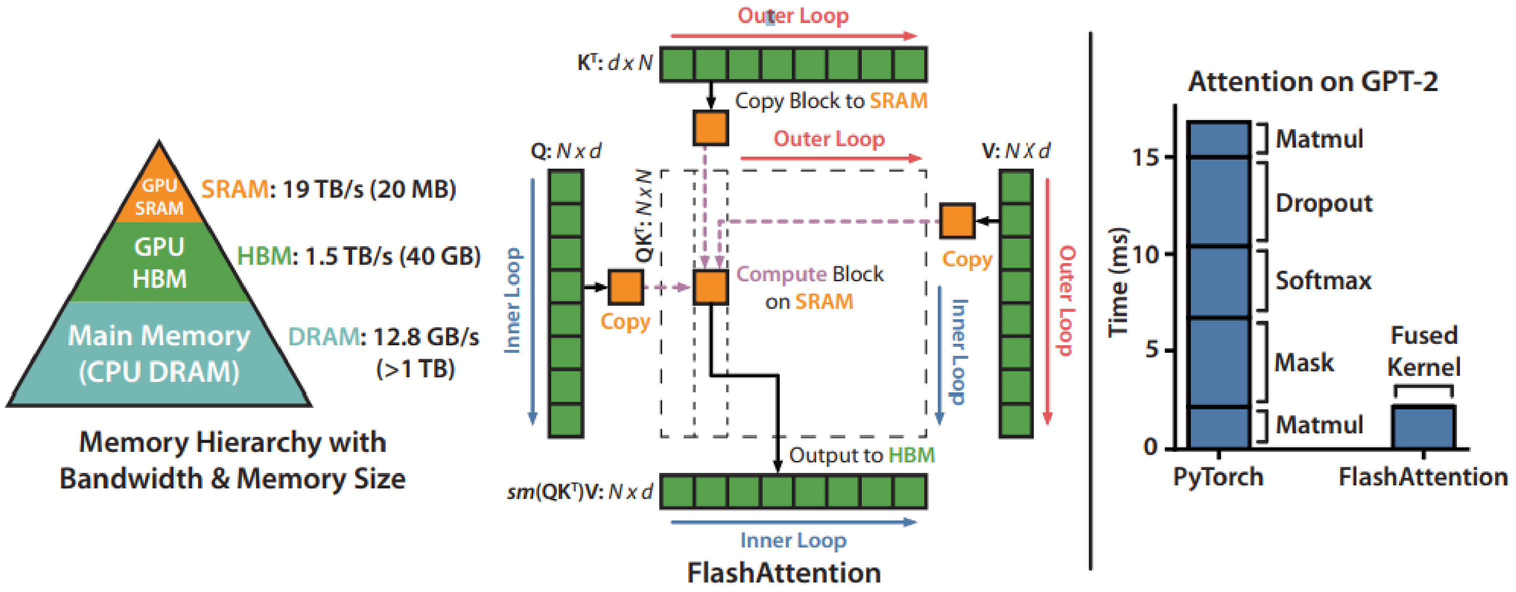
	GPU Type	GPU Power consumption	GPU-hours	Total power consumption	Carbon emitted (tCO_2eq)
OPT-175B	A100-80GB	400W	809,472	356 MWh	137
BLOOM-175B	A100-80GB	400W	1,082,880	475 MWh	183
LLaMA-7B	A100-80GB	400W	82,432	36 MWh	14
LLaMA-13B	A100-80GB	400W	135,168	59 MWh	23
LLaMA-33B	A100-80GB	400W	530,432	233 MWh	90
LLaMA-65B	A100-80GB	400W	1,022,362	449 MWh	173

Touvron et al, 23

Training Foundation Models: **GPU Usage**

Even for each GPU, there's additional considerations

- A little bit of fast memory, lots of slower memory
- Avoid using slow memory when possible
  - FlashAttention: Tiling + computing tricks

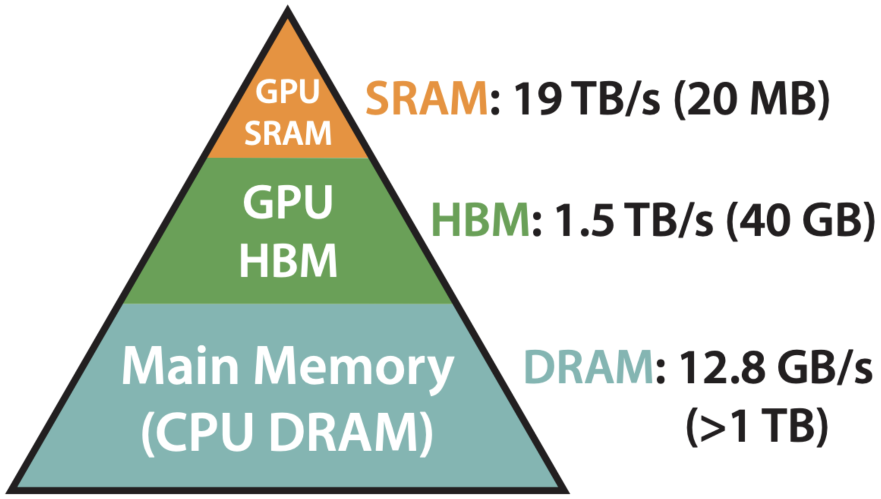


Dao et al '22

Flash Attention

Idea for FlashAttention

- Different kinds of GPU memory



Memory Hierarchy with Bandwidth & Memory Size

  - Fast: on-chip SRAM
    - But very little of this: 192KB for each of ~100 processors for an A100 (20MB)
  - Slow(er): HBM
    - But lots: 40-80GB for an A100

- **Goal**: use fast as much as possible, avoid moving to HBM

Flash Attention: **Basic Idea**

Will use two tricks for higher efficiency

- Tiling and re-computing.

First, recall standard attention

- Will use HBM memory repeatedly
  - Lots of reads and writes:

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Flash Attention: **Tiling**

Will use two tricks for higher efficiency

- Tiling and re-computing.

How do we avoid writing and reading from HBM?

- A: don't load the whole thing, use custom **tiling** and save the pieces (small). Standard version

$$m(x) := \max_i \quad x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \ldots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

- Tiling version: two components (can extend)

$$m(x) = m(\begin{bmatrix} x^{(1)} \ x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} \ x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Flash Attention: **Recomputing**

Will use two tricks for higher efficiency

- Tiling and re-computing.

How do we avoid writing and reading from HBM?

- A: don't load the whole thing, use custom **tiling** and save the pieces

"Tiling enables us to implement our algorithm in one CUDA kernel, loading input from HBM, performing all the computation steps (matrix multiply, softmax, optionally masking and dropout, matrix multiply), then write the result back to HBM (masking and dropout in Appendix B). This avoids repeatedly reading and writing of inputs and outputs from and to HBM."

Don't we need to store full S, P for backwards pass, anyway?

- A: **No!** Can recompute on the fly S, P on the fly

Flash Attention: **Tradeoffs?**

Will use two tricks for higher efficiency

- Tiling and re-computing.

What's the tradeoff?

- Using tiling and computing/re-computing things normally trades off **memory consumption** for **speed**

- **But…** by reducing memory consumption, we can stick to fast memory only
  - And this makes us **much faster**
  - So **no tradeoff** at all (except for needing custom CUDA kernels ☺)

Flash Attention: **Tradeoffs?**

Will use two tricks for higher efficiency

- Tiling and re-computing.

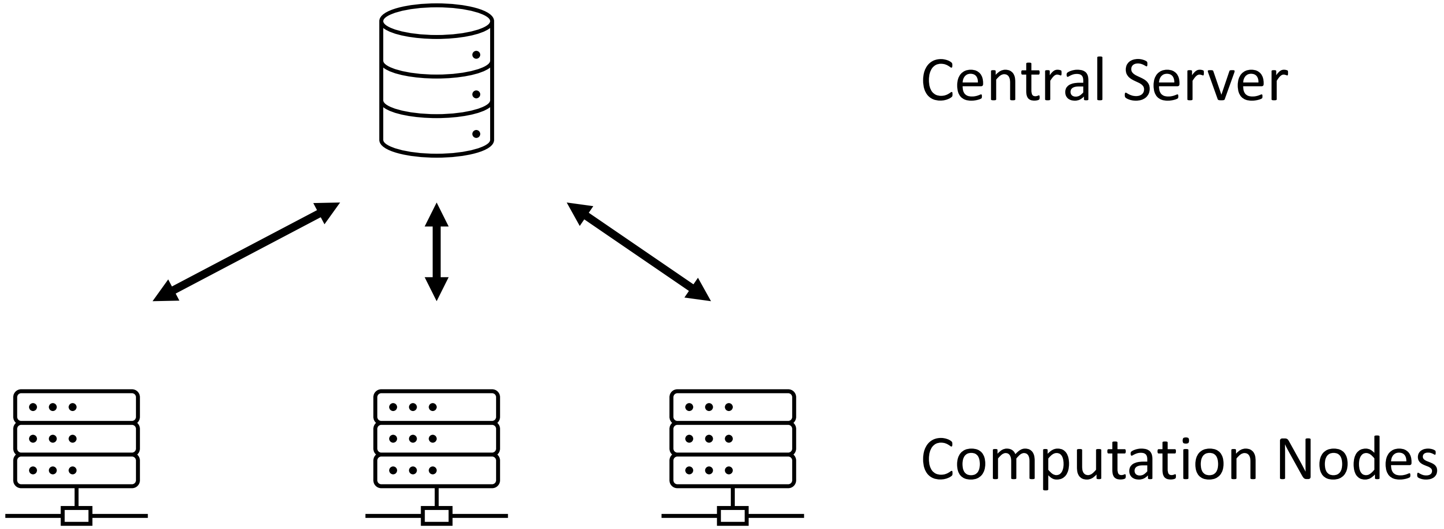
Results:

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Training Foundation Models: **Parallelization**

Traditional approach is to **distribute** training loads

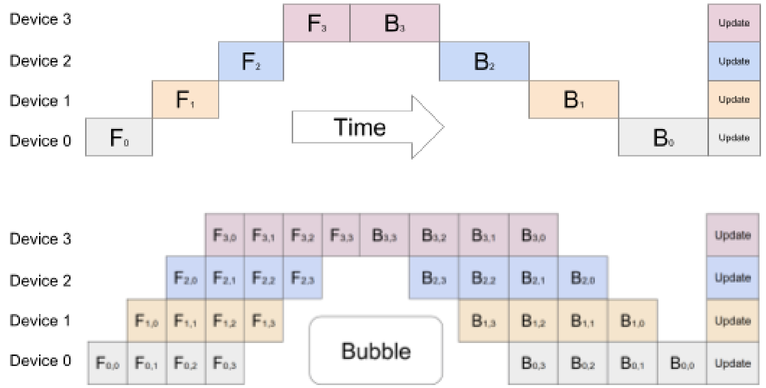
- Classic centralized distributed training
  - Synchronize each local gradient update
  - Send synchronized vector back to each node (lots of communication!)



Training Foundation Models: **Parallelization**

Traditional approach is to **distribute** training loads

- This is by itself impossible (each node *can't* handle full model for large models)
- Need further parallelism:
  - **Data**: each node sees a different slice of data
  - **Weights/tensors**: chunks so no GPU sees whole model
  - **Pipeline**: only a few layers per GPU



Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. *Bottom:* GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.

- Great resource:

https://huggingface.co/blog/bloom-megatron-deepspeed

Break & Questions

Outline

- **Finish Up Last Time**
  - RL training, RLVR, GRPO, datasets
- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Start Efficient Inference**
  - Speculative decoding, early-exit strategies, Flash decoding

Efficient Inference

Similar goal to training

- Gains are more visible

Task	M_q	Temp	γ	α	Speed
EnDe	T5-small ★	0	7	0.75	**3.4X**
EnDe	T5-base	0	7	0.8	2.8X

Leviathan et al '23

Many different approaches. We'll talk about two:

- **Speculative decoding**
  - Inspired by speculative execution in computer architecture
- **Adaptive language modeling**
  - Inspired by early termination methods in ML

Speculative Decoding: Idea

What's slow in autoregressive generation?

- Have to wait for a token to be generated before generating the next token

If we're not *generating*, not slow---can compute probabilities quickly

- Processing the fixed prompt can be reasonably fast

Idea: what if we use a more efficient model for token generation, and then check to see it's OK with original model?

Speculative Decoding: Idea

Idea: what if we use a more efficient model for token generation, and then check to see it's OK with original model?

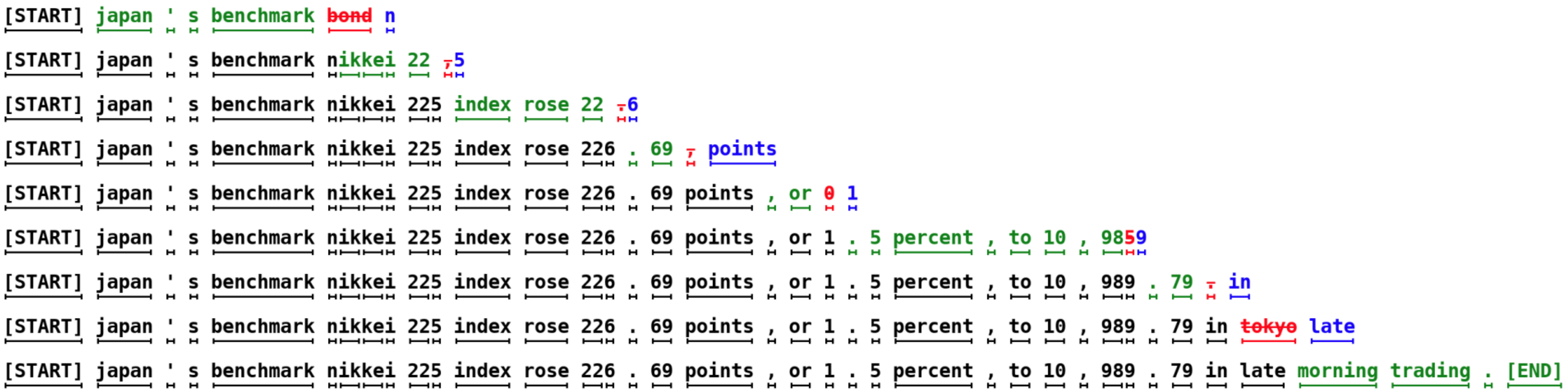
Problem: what if the generated tokens have different probabilities?

- Can reject new ones
- Can run multiple of these in parallel, increase the chances we'll find something we want.

Speculative Decoding: Example

Idea: what if we use a more efficient model for token generation, and then check to see it's OK with original model?

- Green: accepted, red: rejected, blue: original LM.
  - Each line is one iteration of speculative decoding.



Leviathan et al '23

Speculative Decoding: Algorithm

Algorithm:

- M_p original model, M_q small model (efficient)
- Generate γ parallel paths with M_q
- Check what was accepted
  - Adjust if needed
  - Sample from "adjusted" distribution
- Generate one more token from M_p

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\dots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
    $M_p(prefix), \dots, M_p(prefix + [x_1, \dots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \dots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \dots, x_n, t]$

Speculative Decoding: Results

Some sample results:

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X

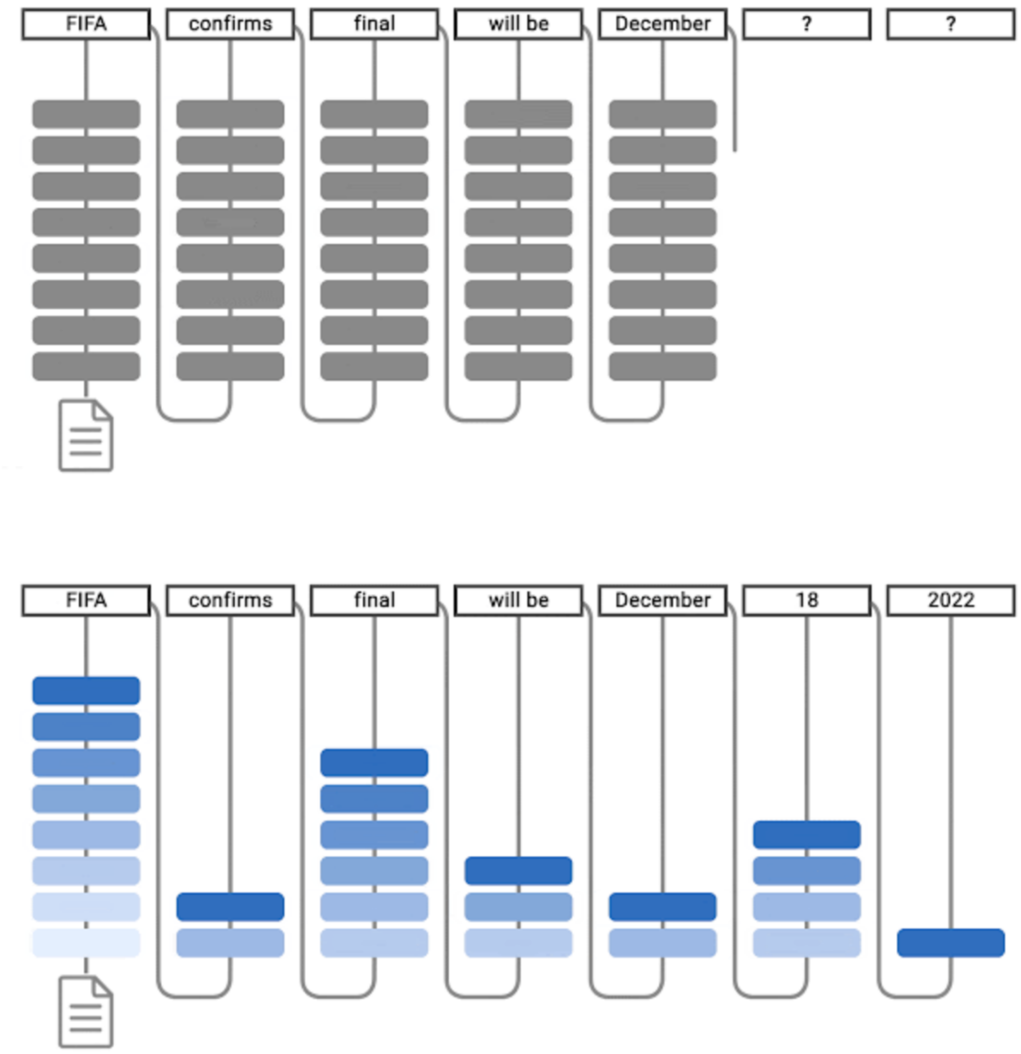
Note: lots of extensions!

- What kind of generation ”paths” should we use?

Adaptive Language Modeling

Basic idea: make predictions based on earlier layers

- When it is safe to do so.
- Goal: introduce constraints and ensure these are satisfied,
  - Textual consistency
  - Risk consistency



Schuster et al ‘22

Thank You!