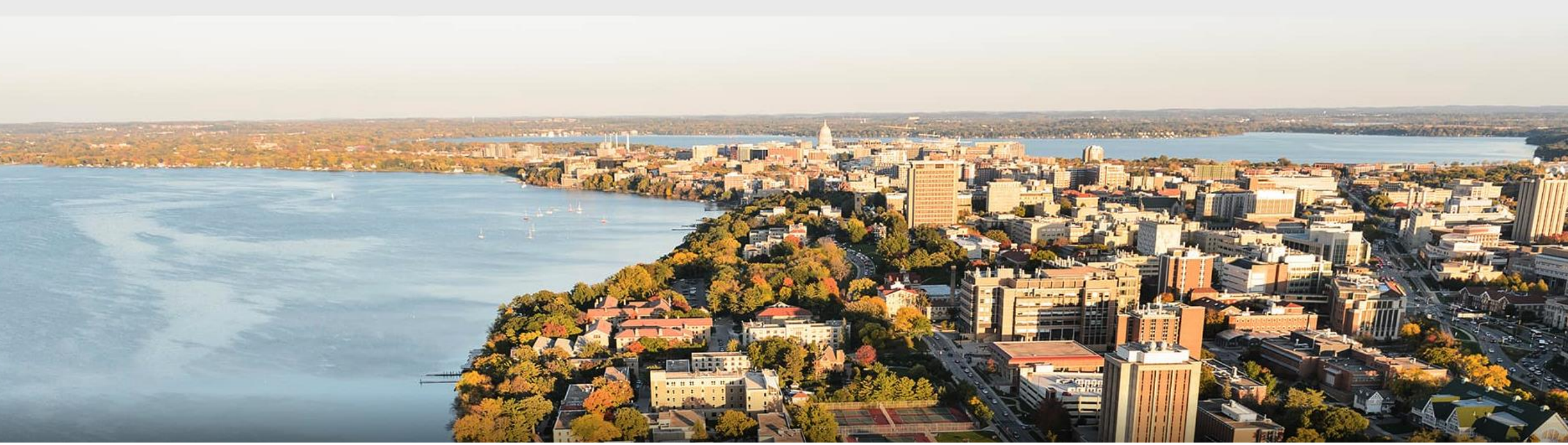


# CS 839: Foundation Models
# **Efficient Training & Inference**

Fred Sala

University of Wisconsin-Madison

**Oct. 15, 2024**

# Announcements

- **Logistics:**
  - Homework 2 in progress.
  - **No OH on Thursday!**
    - Mini-OH at end of lecture today
  - Project information coming out shortly.
- Class roadmap:

| | |
|---|---|
| Tuesday Oct. 15 | Efficient Training & Inference |
| Thursday Oct. 17 | Hybrid Models (**Guest Lecture**) |
| Tuesday Oct. 22 | Data |
| Thursday Oct. 24 | Evaluation |
| Tuesday Oct. 29 | Multimodal Models |

# Announcements

- **Logistics:**
  - Thursday's guest lecture: **Nick Roberts** (UW-Madison, Meta, Together.ai)

  - **Topics**: "Hybrid" architectures, models and neural architecture search.

# Outline

- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Efficient Inference**
  - Speculative decoding, early-exist strategies, Flash decoding
- **Mini-OH**
  - Also form groups

# Outline

- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Efficient Inference**
  - Speculative decoding, early-exist strategies, Flash decoding
- **Mini-OH**
  - Also form groups

# 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"*

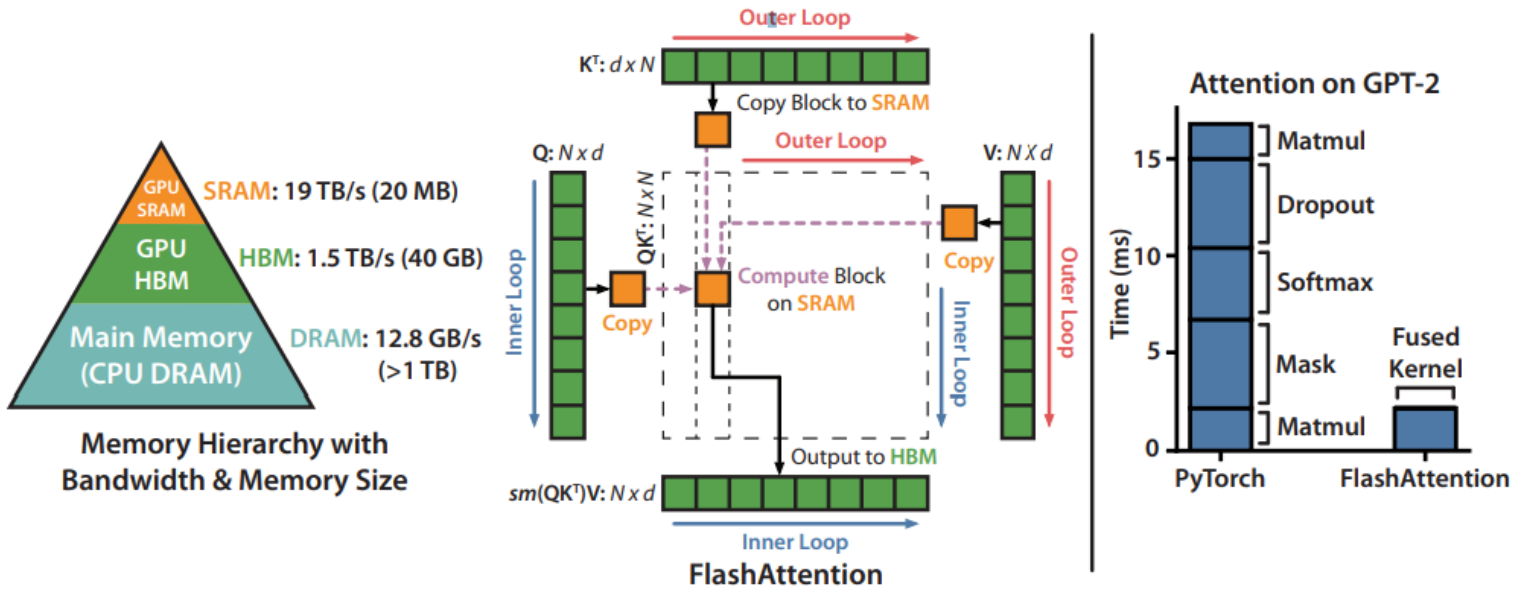
| | GPU Type | GPU Power consumption | GPU-hours | Total power consumption | Carbon emitted ($tCO_2eq$) |
|---|---|---|---|---|---|
| OPT-175B | A100-80GB | 400W | 809,472 | 356 MWh | 137 |
| BLOOM-175B | A100-80GB | 400W | 1,082,880 | 475 MWh | 183 |
| LLaMA-7B | A100-80GB | 400W | 82,432 | 36 MWh | 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

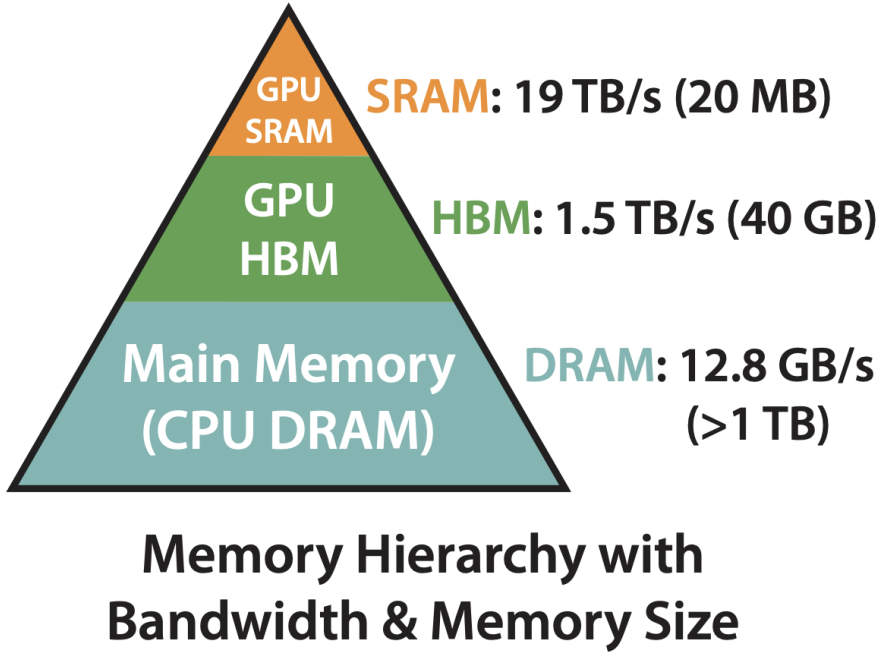



Dao et al '22

# 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

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

- Tiling version: two components (can extend)

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

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

# 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.

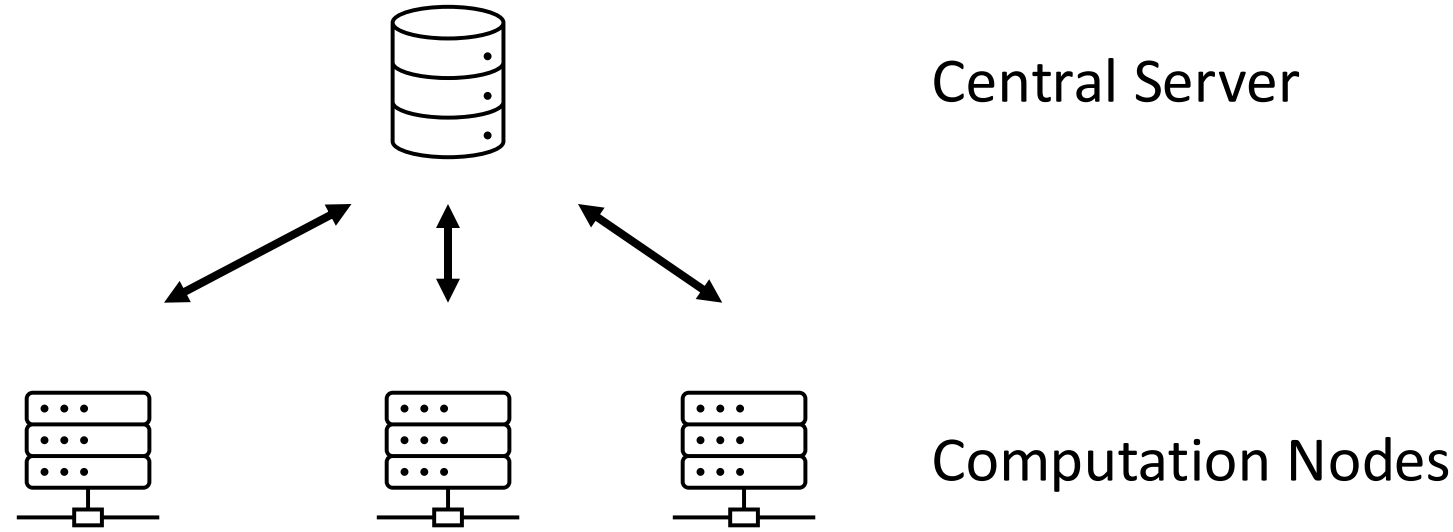
Results:

| 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!)



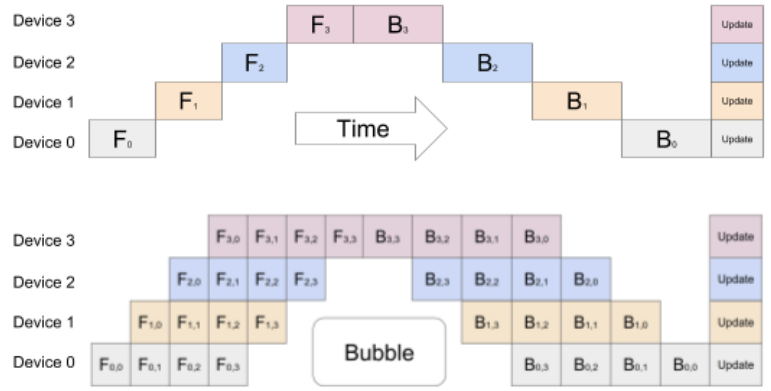
# Training Foundation Models: **Parallelization**

Traditional approach is to **distribute** training loads

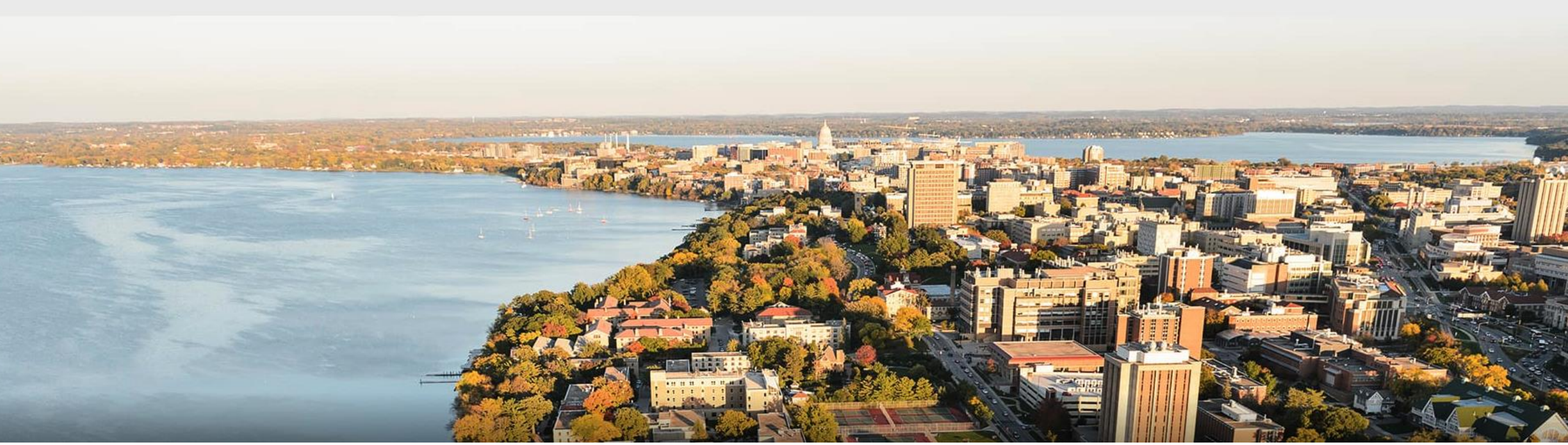
- This is by itself impossible (each node *can't* handle full model for large models)
- Need further parallelism:
  - **Data**: each node sees a different slice of data
  - **Weights/tensors**: chunks so no GPU sees whole model
  - **Pipeline**: only a few layers per GPU

- Great resource:

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




*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.*



# Break & Questions

# Outline

- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Efficient Inference**
  - Speculative decoding, early-exist strategies, Flash decoding
- **Mini-OH**
  - Also form groups

# Efficient Inference

Similar goal to training

- Gains are more visible

| TASK | $M_q$ | TEMP | $\gamma$ | $\alpha$ | 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.

```
[START] japan ' s benchmark ~~bond~~ n
[START] japan ' s benchmark nikkei 22 ~~,~~5
[START] japan ' s benchmark nikkei 225 index rose 22 ~~.~~6
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 ~~,~~ points
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 0 1
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 ~~,~~ in
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in ~~tokyo~~ late
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

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 $\gamma$ guesses $x_{1,\ldots,\gamma}$ from $M_q$ autoregressively.
**for** $i = 1$ **to** $\gamma$ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
**end for**
▷ Run $M_p$ in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses $n$.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ 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** $prefix + [x_1, \ldots, x_n, t]$

# **Speculative Decoding:** Results

Some sample results:

| TASK | $M_q$ | TEMP | $\gamma$ | $\alpha$ | 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 |

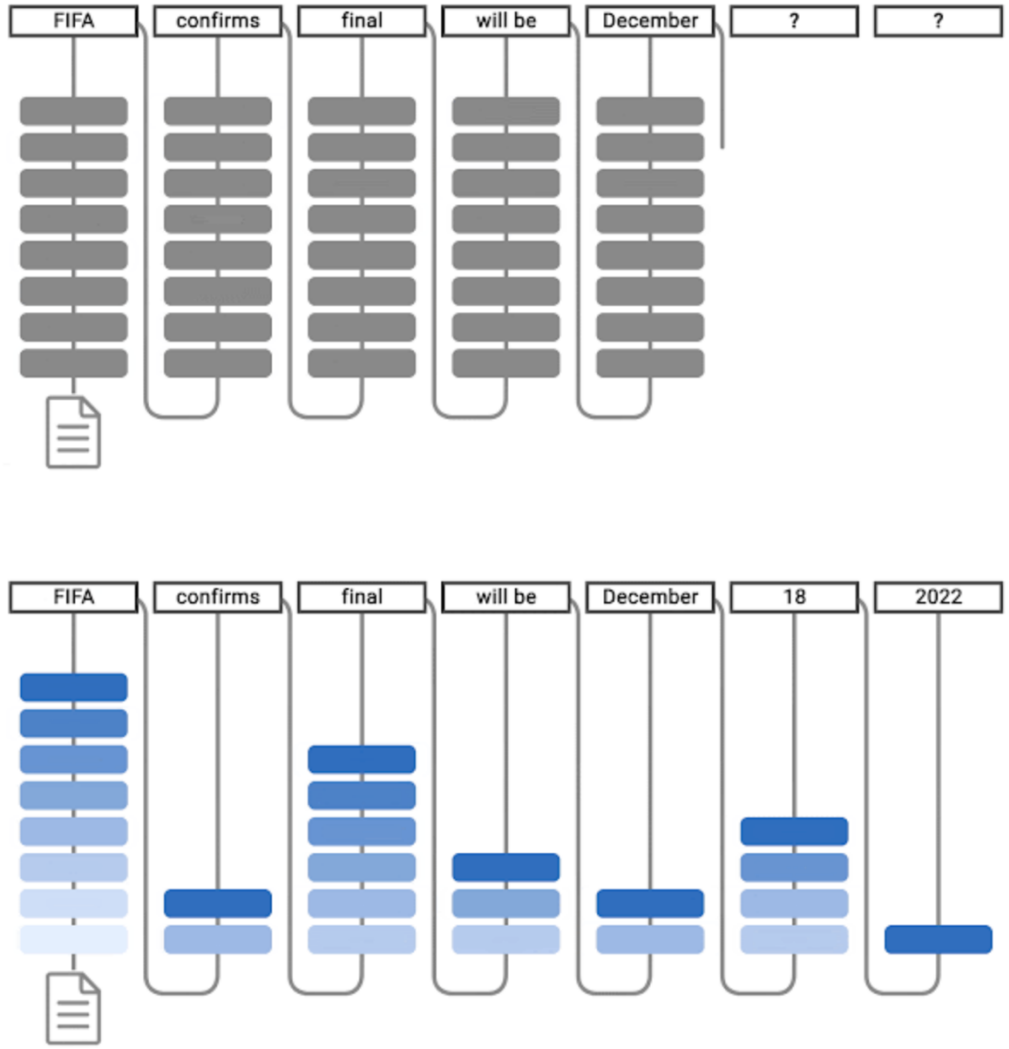
Note: lots of extensions!

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

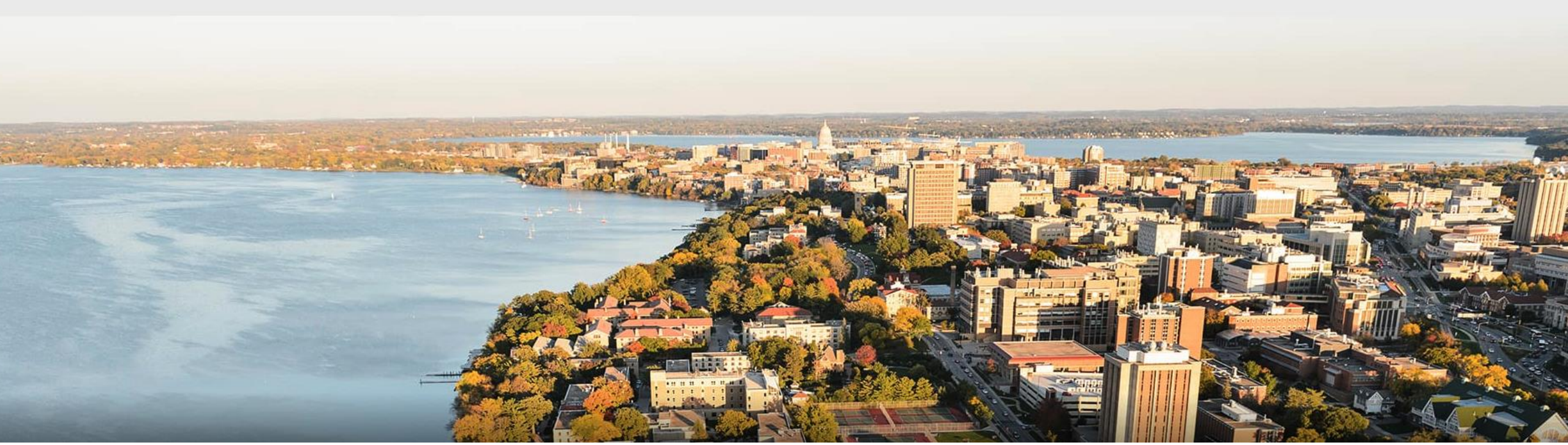
# Adaptive Language Modeling

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



# Break & Questions

# Outline

- **Efficient Training**
  - Scale, memory optimization (FlashAttention), parallelism, heterogenous training
- **Efficient Inference**
  - Speculative decoding, early-exist strategies, Flash decoding
- **Mini-OH**
  - Also form groups

# Thank You!