Efficient and Elastic LLMs

Prateek Jain
Google Research India

Based on joint works with:

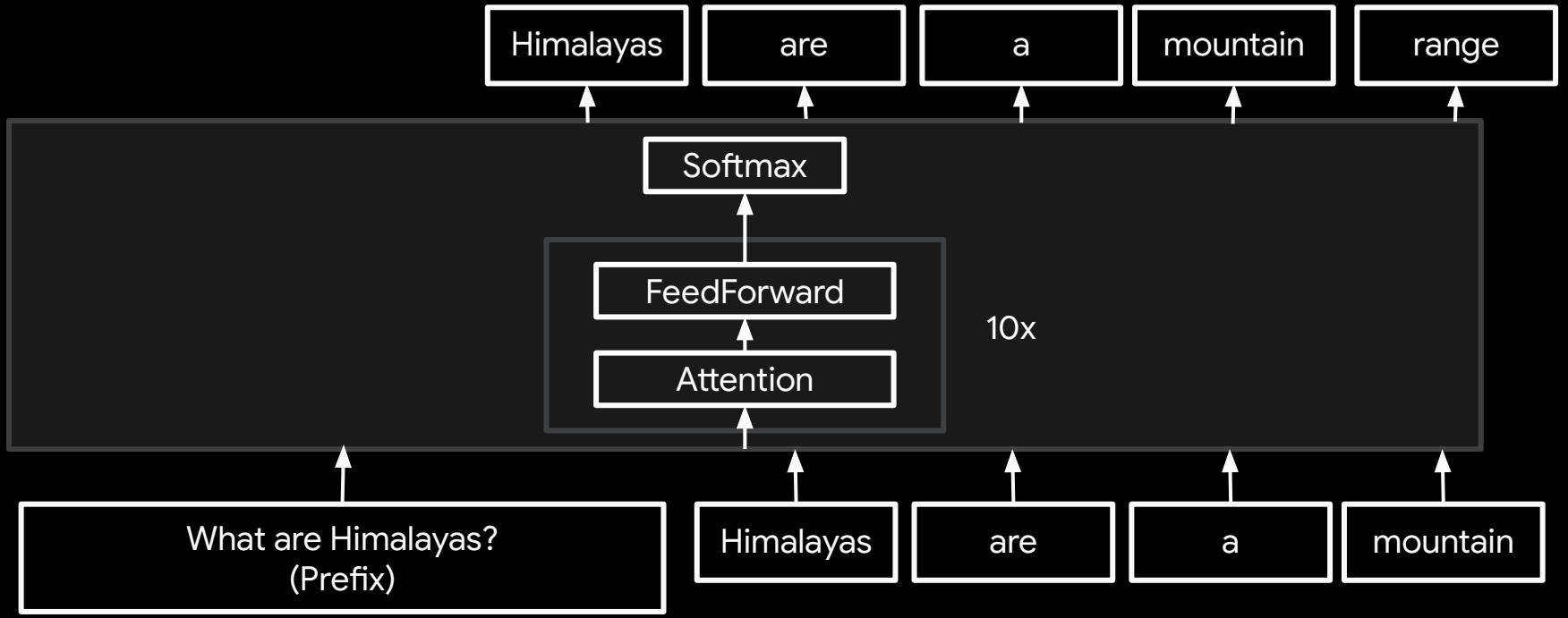
Aishwarya PS, Yashas Samaga, Varun Yerram, Pranav Nair,
Chong You, Srinadh Bhojanapalli, Toby Boyd,
Devvrit Khatri, Sneha Kudugunta, Aditya Kusupati,
Tim Dettmers, Hannaneh Hajishirzi, Yulia Tsvetkov, Kaifeng Chen,
Inderjit Dhillon, Sham Kakade, Ali Farhadi, Sanjiv Kumar, and Praneeth Netrapalli

Why care about inference latency?

- Training is a one time cost. Inference is a continuous cost.
- [1] estimates the carbon emissions (and consequently, cost) of inference to be about 80% over the lifetime of a model.
- High inference cost is the biggest hurdle to deployment of most capable models today.
- Very active area of research with several new ideas across algorithms, architectures, systems, hardware etc.

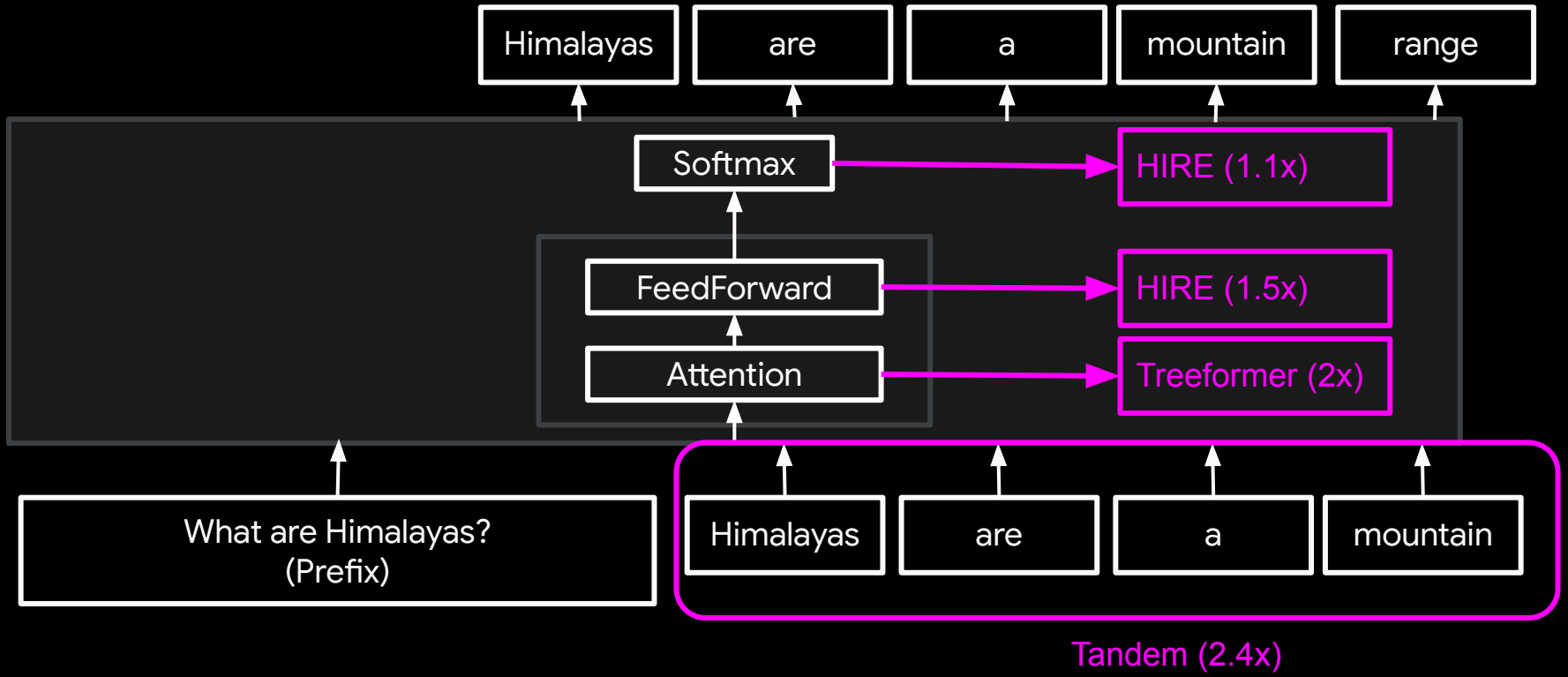
[1] Carbon emissions and large neural network training by Patterson et al. 2021

Inference latency of Generative LLMs

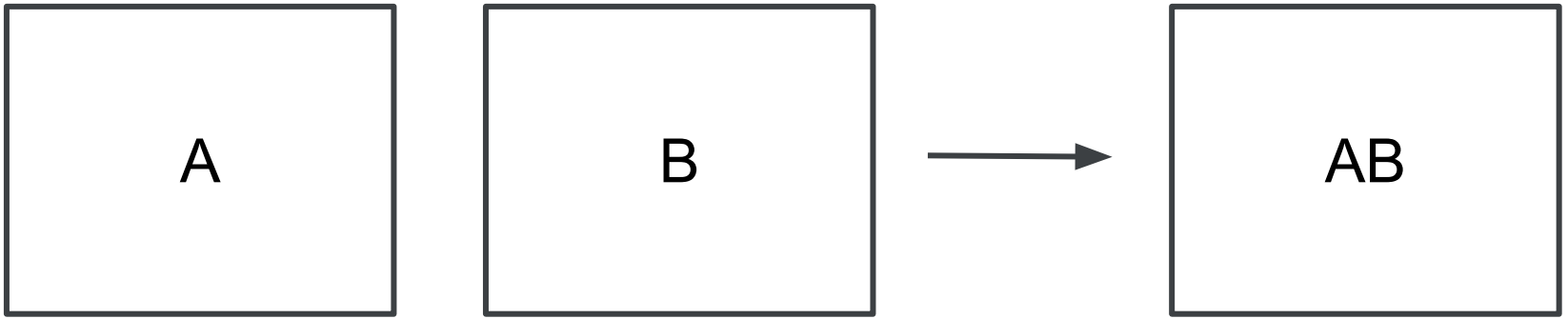


- Autoregressive response generation is the key bottleneck.
- Within the model, there are different components: *Attention*, *FeedForward* and *Softmax* layers.

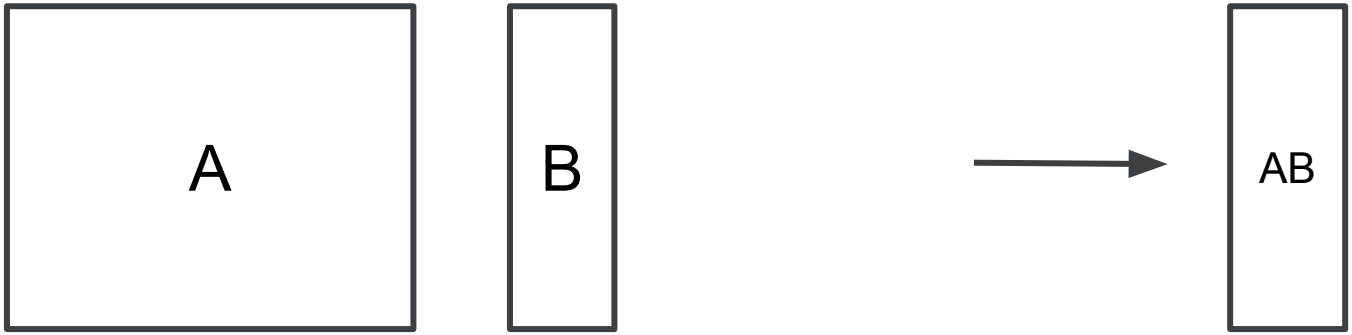
Inference latency of Generative LLMs



Matrix-matrix vs matrix-vector multiplications



Takes the same amount of time as

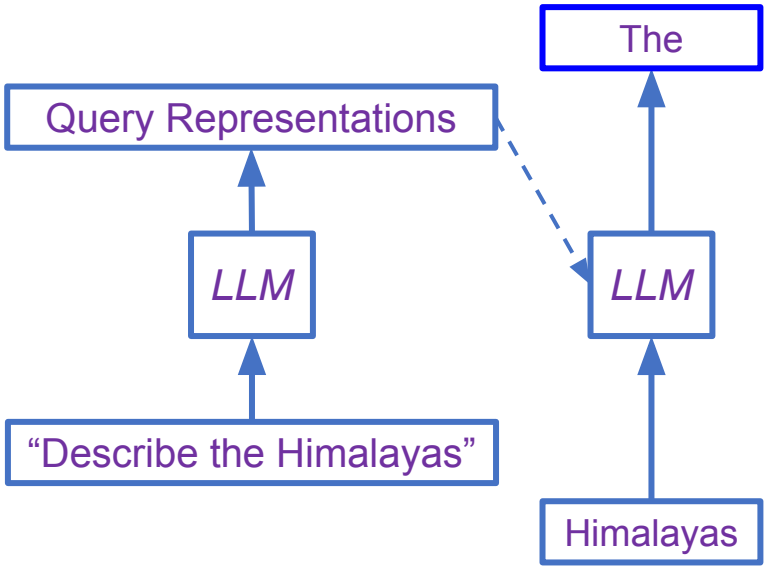


Part I: Tandem Transformers

Background: **Autoregressive** LLM inference

Feed forward + Attention

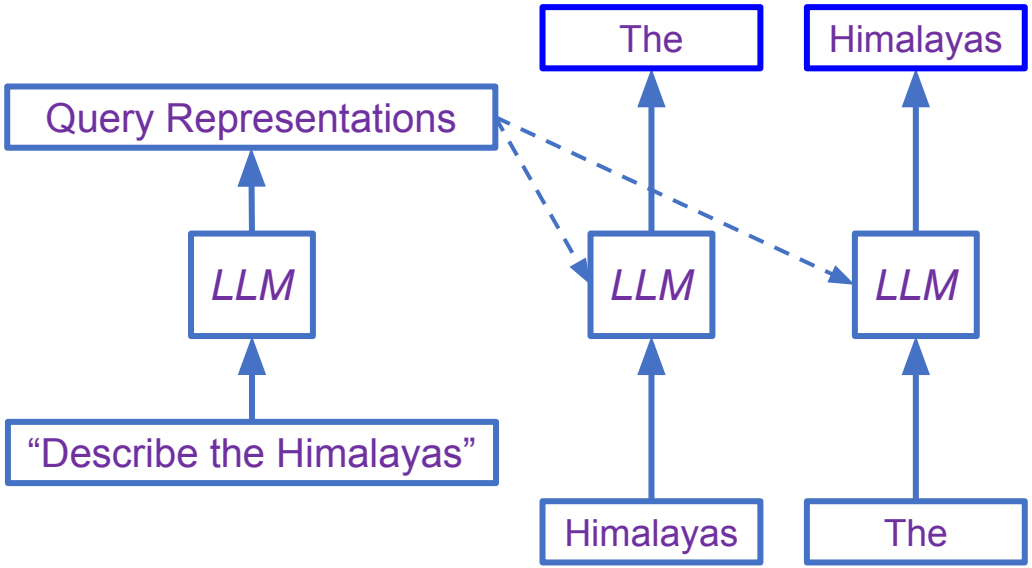
Attention only



Background: **Autoregressive** LLM inference

Feed forward + Attention

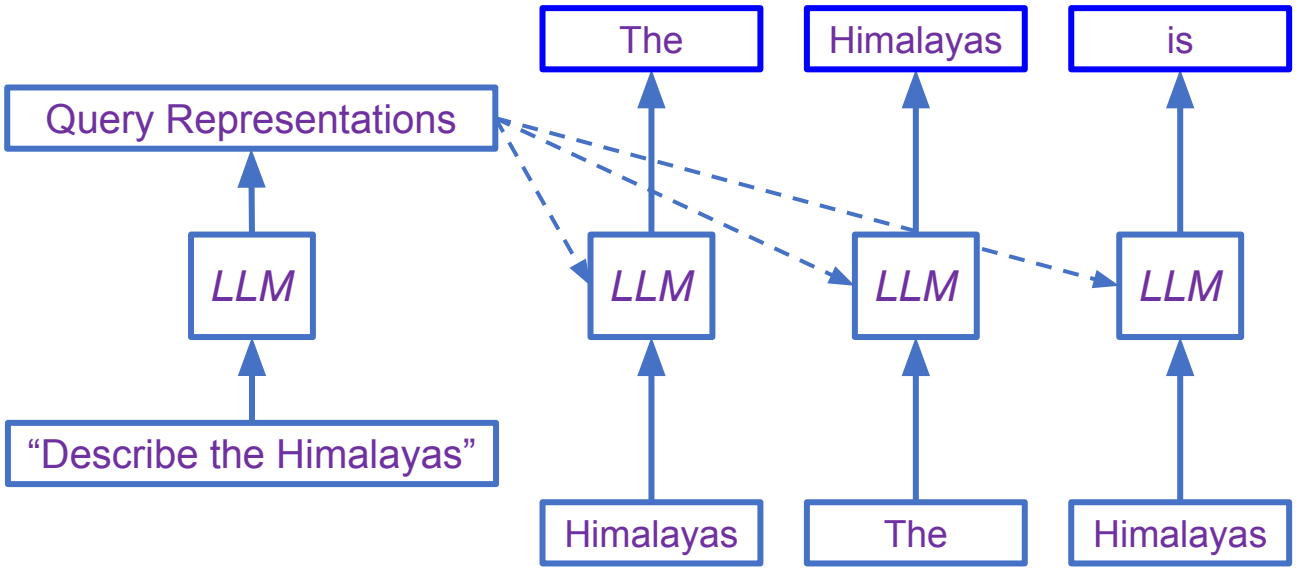
Attention only



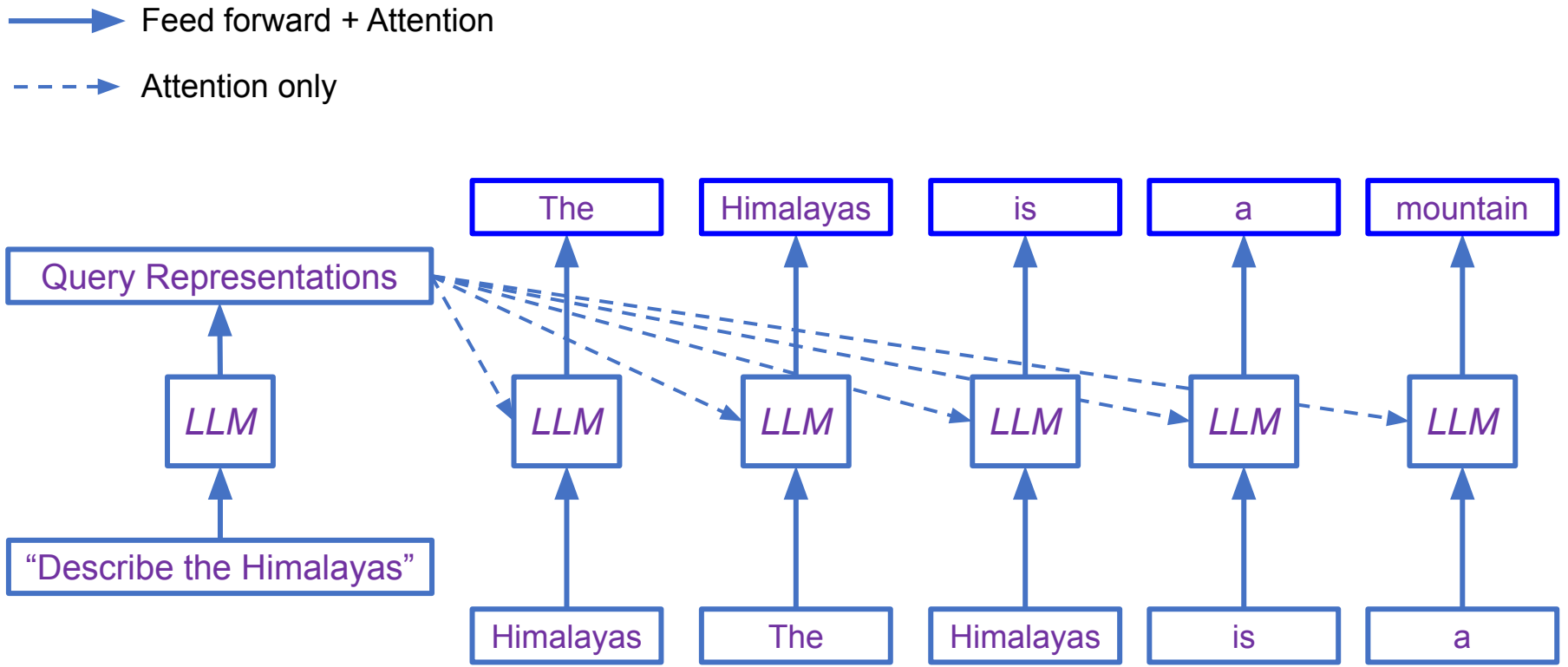
Background: **Autoregressive** LLM inference

Feed forward + Attention

Attention only



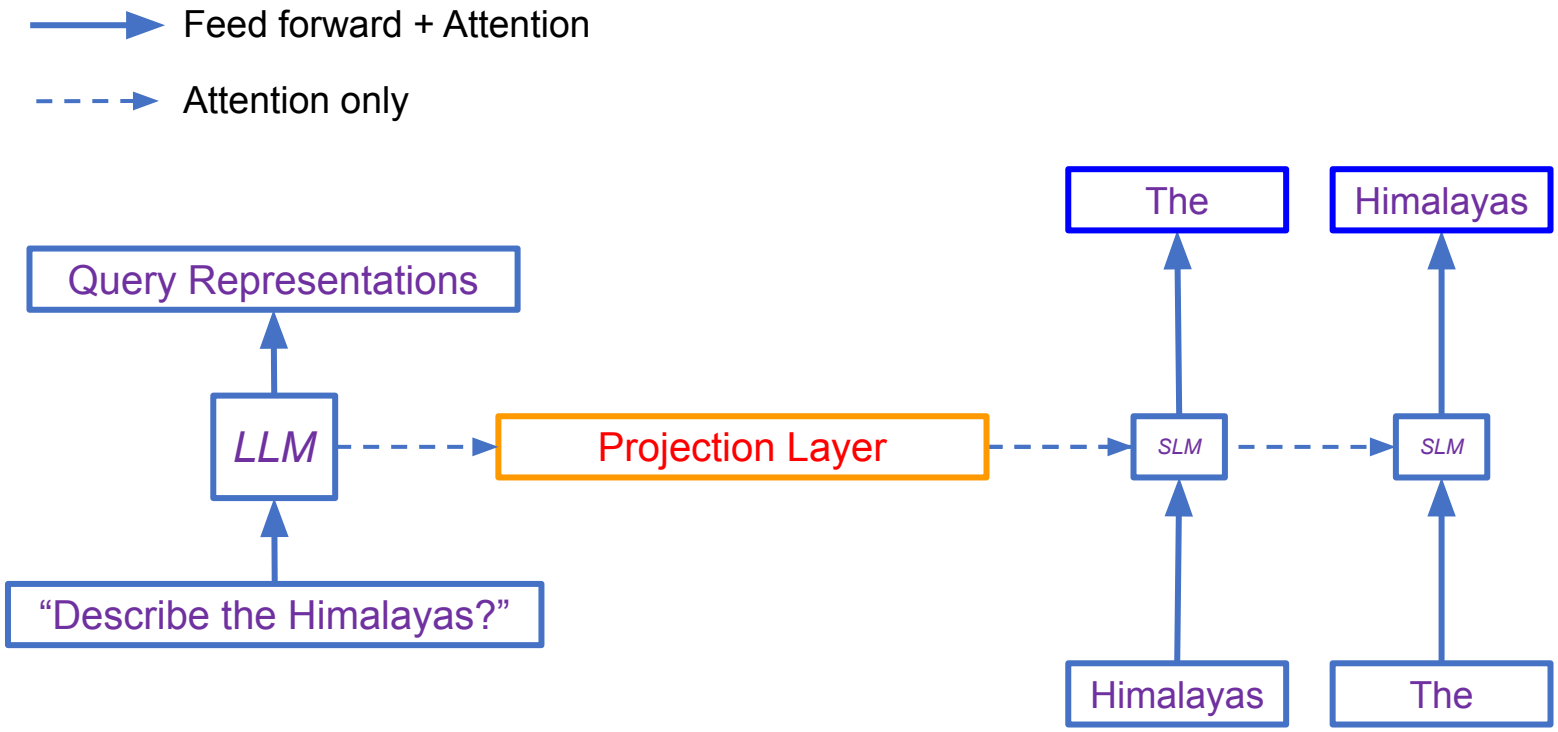
Background: **Autoregressive** LLM inference



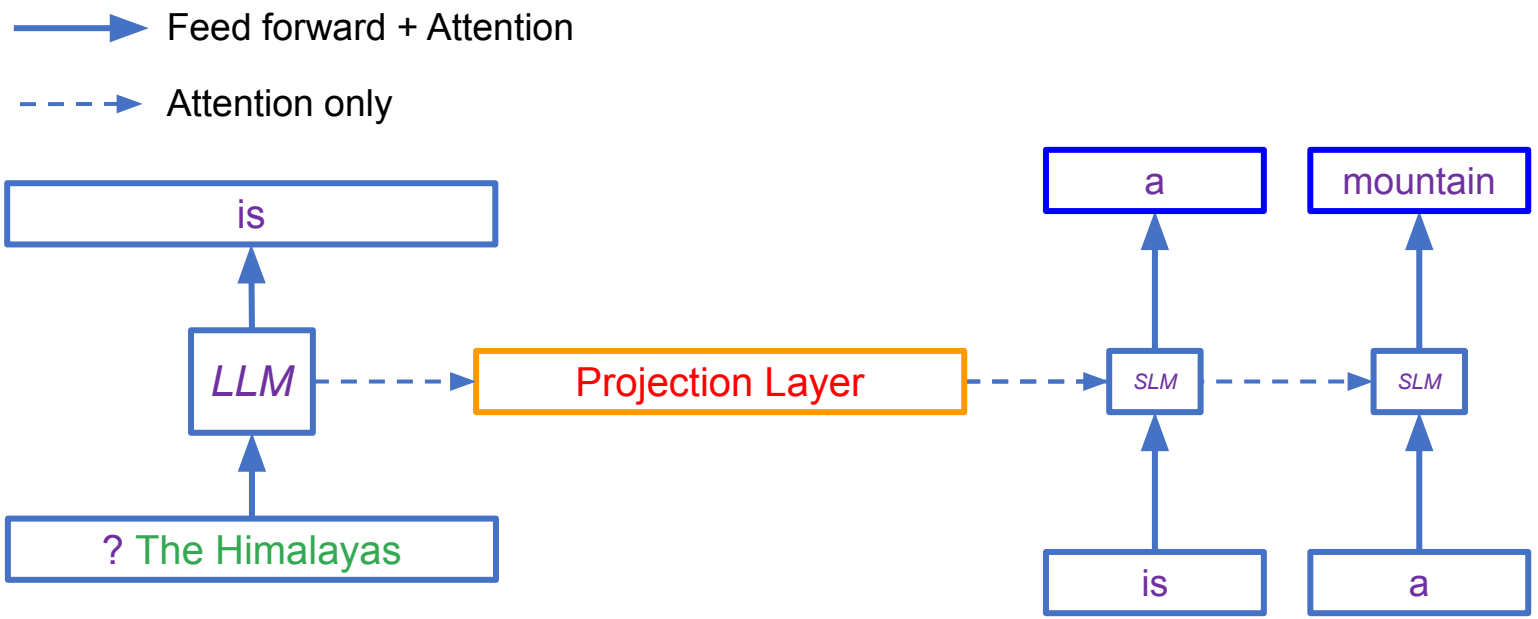
Research questions

- Autoregressive generation leads to poor utilization of GPUs/TPUs since we perform matrix-vector multiplications instead of matrix-matrix multiplications.
  - Can we mitigate the autoregressive component?
- Two key conceptual tasks of an LLM: **understanding** and **generation**.
  - Classically, encoder-decoder models decouple these and often use **larger encoder** models (for understanding) with **smaller decoder** models (for generation).
  - Decoder-only architecture couples understanding and generation.
  - Can we decouple capacity required for these?

Tandem Transformers (Block length: 2)



Tandem Transformers (Block length: 2)



Questions:

1. Can SLM adapt to LLM's representations?
2. Can LLM's representations help SLM become more powerful?

Tandem model training

- We train a tandem model with LLM = PaLM2-Bison and SLM = PaLM2-Gecko
- In terms of size, PaLM2-Gecko < PaLM2-Otter < PaLM2-Bison
- After initializing with pretrained models, freeze LLM and train only the projection layers and SLM.
- Tandem-CE: trained with CE loss wrt ground truth labels
- Tandem-Distill: continue training Tandem-CE also with CE wrt PaLM2-Bison output logits.
- PaLM2-Gecko-Distill: continue training PaLM2-Gecko with distillation loss as above.

Tandem Transformers: Evaluation

Benchmark/Method	Primary (P)	Secondary (S)	**Tandem (P+S)**
GPT3-Gen	57.5	28.8	**44.0**
GPT3-Rank	73.6	57.1	**70.2**
TydiQA-GoldP	73.4	55.0	**69**
Super-GLUE	81.4	62.8	**78.8**

Speed-up over Primary: **2.74x**

Pretraining evaluation

	PaLM2-Gecko	PaLM2-Gecko-Distil	Tandem-CE **(ours)**	Tandem-Distil **(ours)**
Accuracy (ground truth)	55.06	56.50	58.35	**58.61**
CE loss (ground truth)	2.14	2.12	**1.94**	1.99
Relative accuracy	74.64	75.30	80.00	**81.00**
Relative TV distance	0.391	0.318	0.178	**0.141**

Table 1. Accuracy and cross entropy (CE) loss of Tandem transformers with respect to ground truth labels as well as the predictions of the primary model $\mathcal{M}_L$, PaLM2-Bison. As is clear from the results, the Tandem model of PaLM2-Gecko and PaLM2-Bison substantially outperforms the stand alone PaLM2-Gecko model.

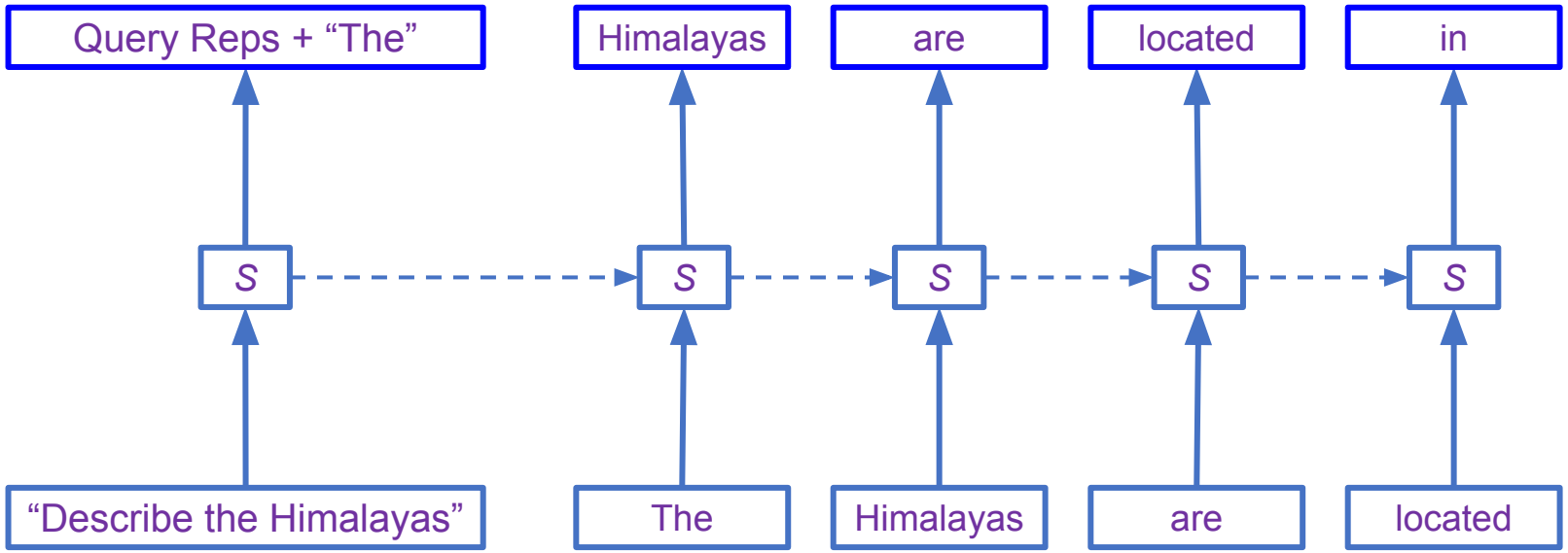
Downstream evaluations

Dataset	PaLM2-Gecko	Tandem-CE **(ours)**	Tandem-Distil **(ours)**	PaLM2-Otter	PaLM2-Bison
Generative-tasks	28.8	37.1	44.0	51.1	57.5
MBPP	4.8	13.8	21.2	20.8	30.4
WMT22-1shot-to-nonenglish	35.1	37.4	44.1	48.4	50.5
TydiQA-GoldP	55.0	65.7	69.0	69.7	73.4
Super-GLUE	62.8	78.5	78.8	79.0	81.5
Speedup over PaLM2-Bison	6.397×	2.744×	2.744×	2.359×	1×

Table 3. Standalone evaluation of the Tandem model. The first five rows present downstream evaluations of the Tandem transformers on a variety of generative and ranking tasks. We see that the Tandem model substantially improves upon the performance of stand alone PaLM2-Gecko model, and is on par with the PaLM2-Otter model. On the other hand, the latency evaluations in the last row demonstrate that the Tandem model is about 1.16x faster than the PaLM2-Otter model.

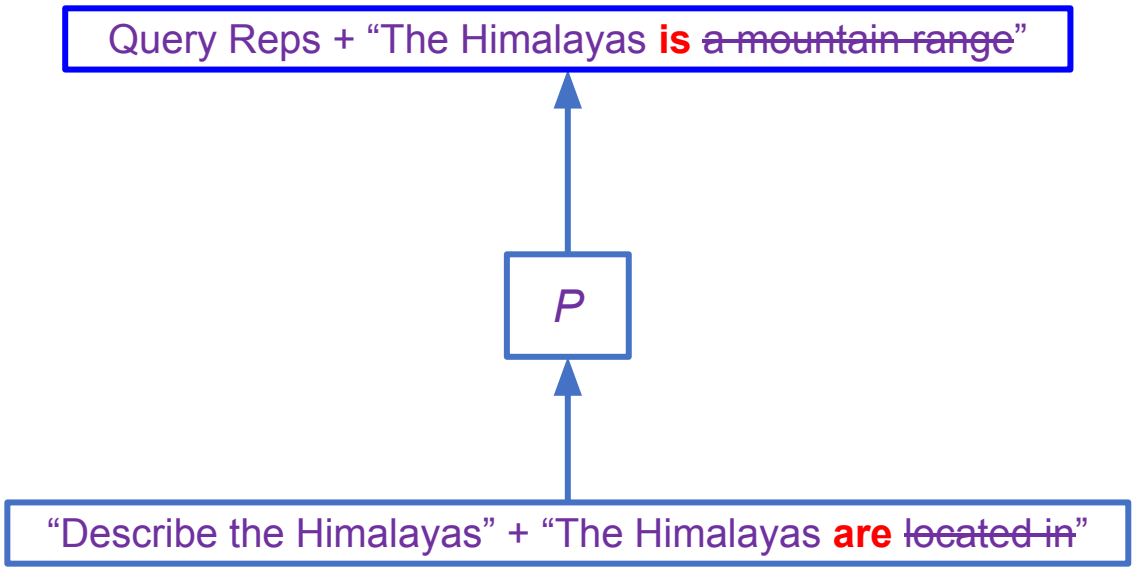
Can we use tandem without worrying about accuracy?

SPEED - Speculative Decoding [1]



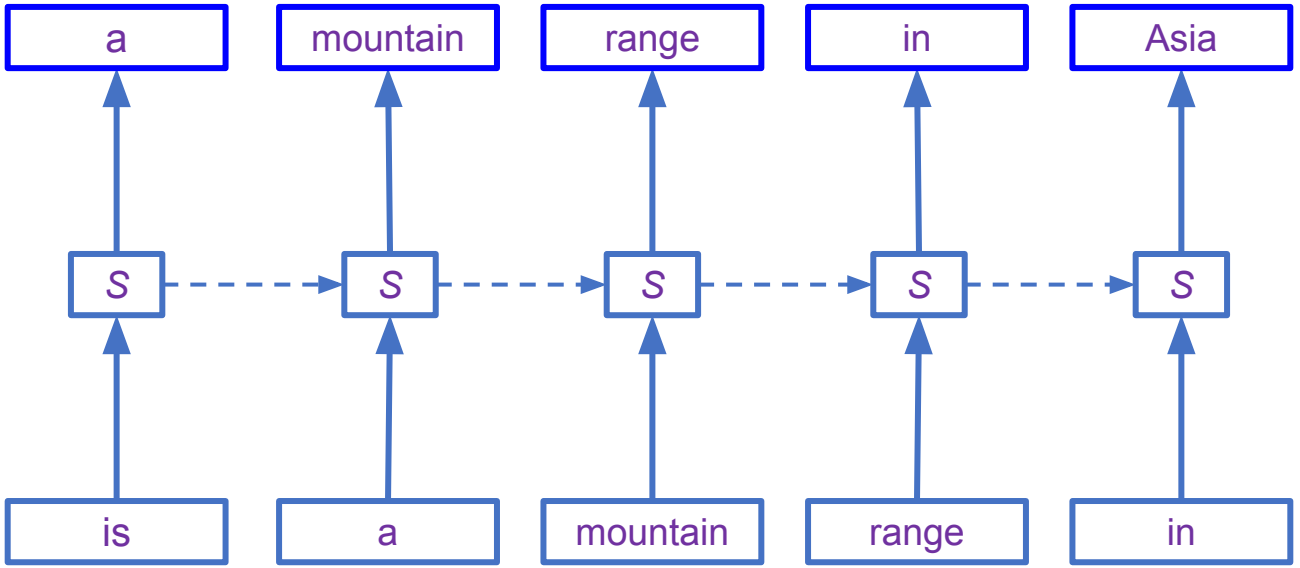
[1] Fast inference from transformers via speculative decoding by Leviathan et al. ICML 2023

SPEED



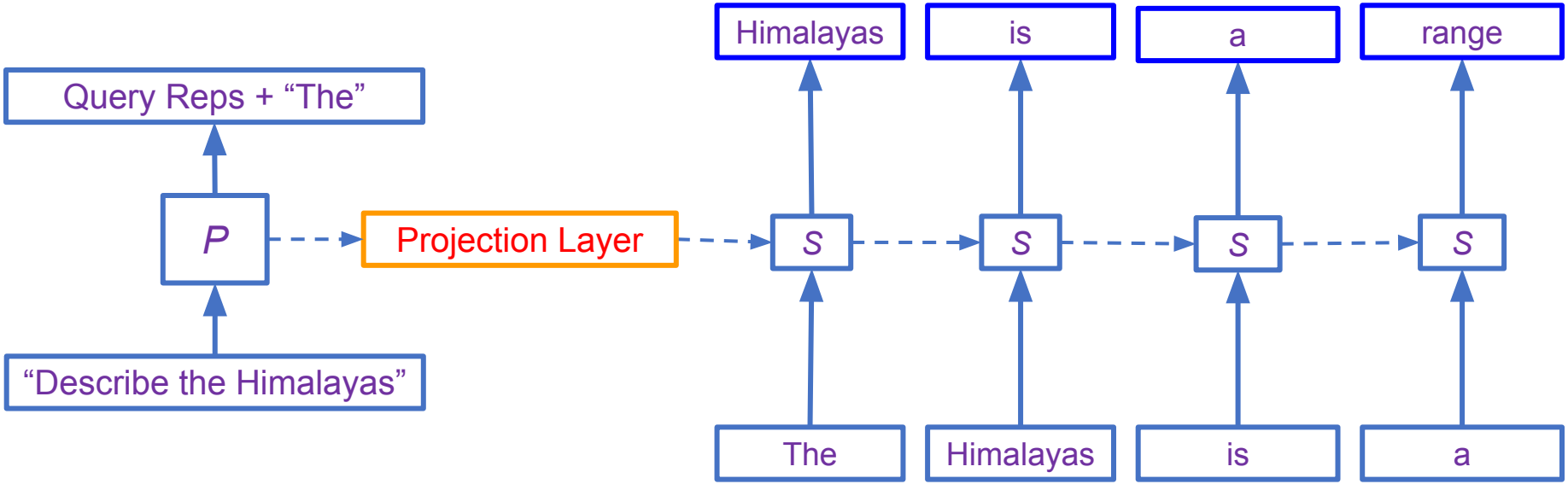
are ≠ is

SPEED

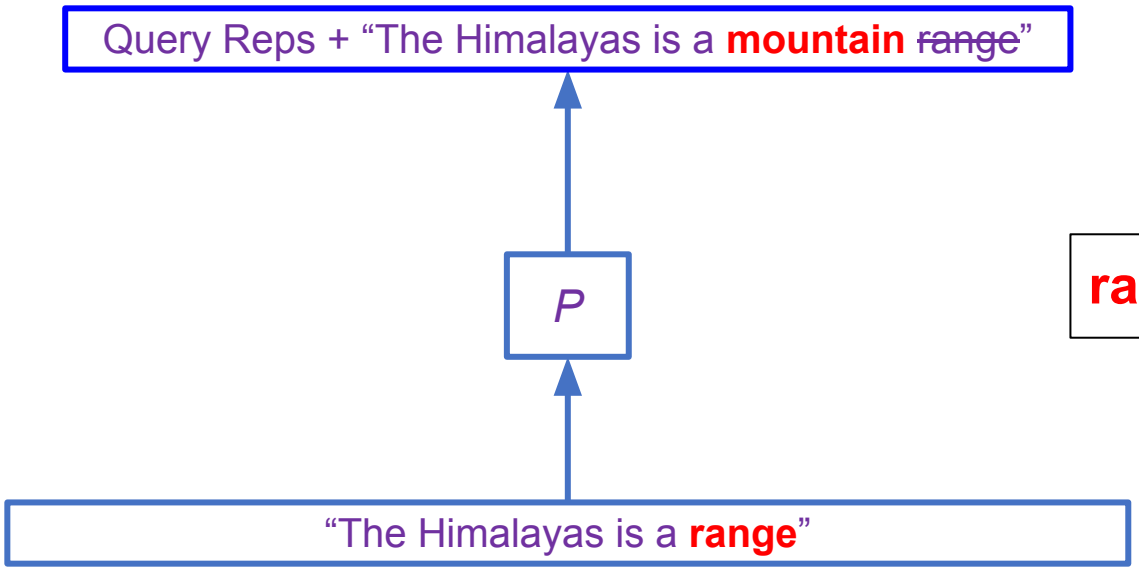


Less backtracking → More speedup.

SPEED+Tandem



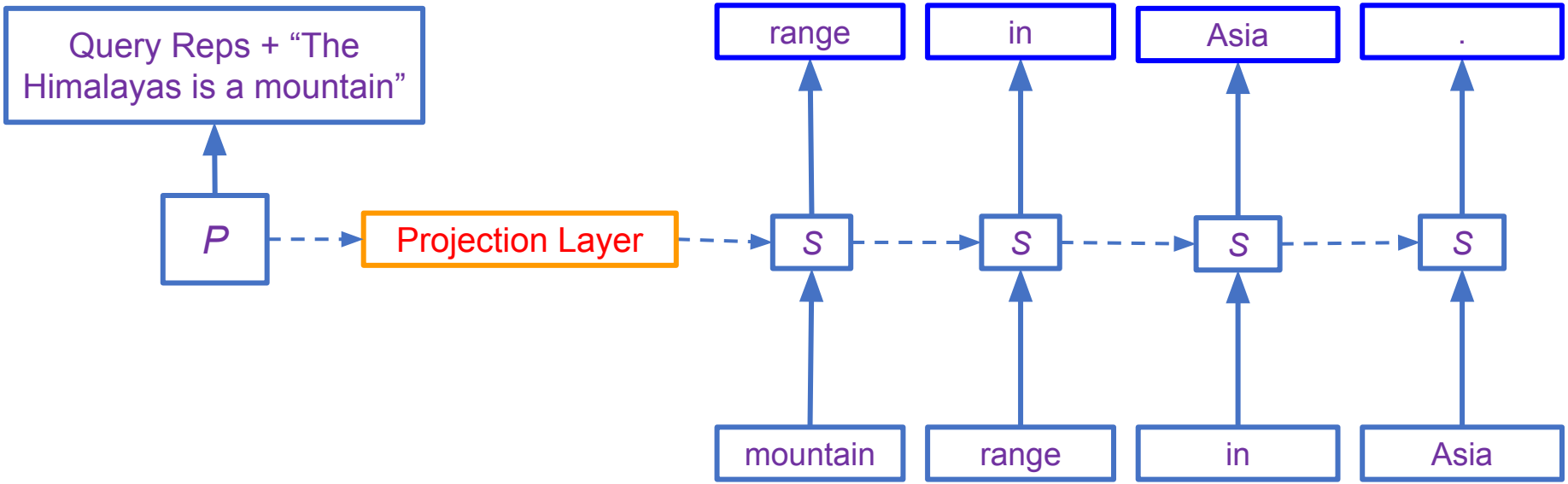
SPEED+Tandem



range ≠ mountain

SPEED+Tandem

P = PaLM2-Bison
S = PaLM2-Gecko



Related works for better secondary models in SPEED

- [1] Parallel decoding – attaching more heads to the primary model to predict several tokens ahead.
- [2] Drafting based on retrieval rather than through a SLM.
- The drafters in both above setups are not powerful enough for difficult generation tasks.
- [3] Distilling the drafter model with primary model's logits is more effective.
    - We use this for comparison.

[1] Blockwise parallel decoding for deep autoregressive models by Stern et al. NIPS 2018
[2] Rest: Retrieval-based speculative decoding by He et al. 2023
[3] Distillspec: Improving speculative decoding via knowledge distillation by Zhou et al. ICLR 2024

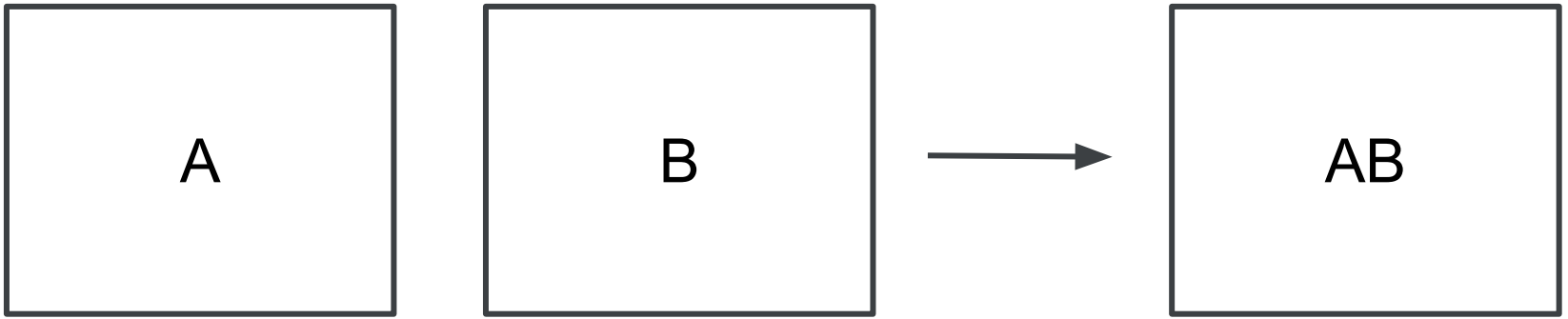
Latency improvements within SPEED

Dataset	Num-Samples	PaLM2-Gecko-Distil (baseline)	Tandem-Distil (ours)	Tandem-Distil (ours; relative gain)
Reddit	1	$\times$ 2.169 ($\gamma = 7$)	2.471$\times$ ($\gamma = 7$)	**1.139**$\times$
	4	$\times$ 1.919 ($\gamma = 5$)	2.234$\times$ ($\gamma = 7$)	**1.164**$\times$
CNN/DailyMail	1	$\times$ 2.219 ($\gamma = 7$)	2.473$\times$ ($\gamma = 7$)	**1.115**$\times$
	4	$\times$ 1.940 ($\gamma = 5$)	2.190$\times$ ($\gamma = 7$)	**1.129**$\times$
LM1B	1	$\times$ 2.348 ($\gamma = 7$)	2.610$\times$ ($\gamma = 7$)	**1.112**$\times$
	4	$\times$ 2.011 ($\gamma = 5$)	2.359$\times$ ($\gamma = 7$)	**1.173**$\times$

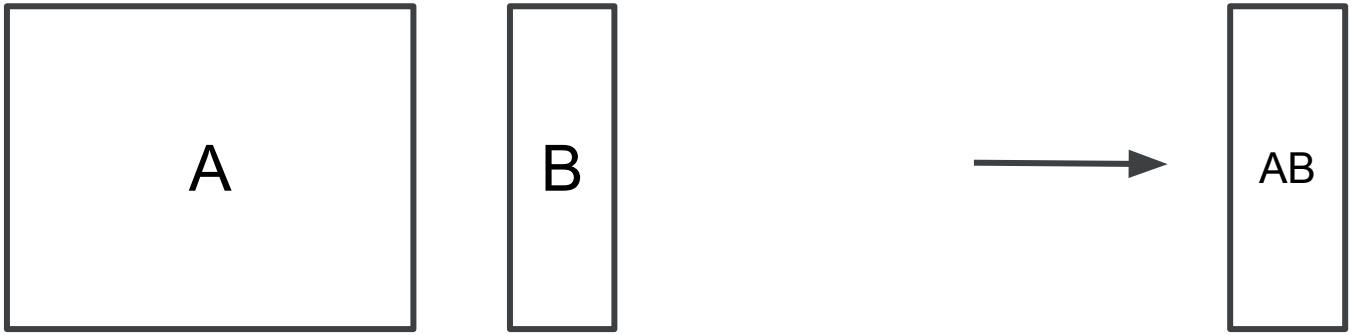
- Speedup wrt stand alone PaLM2-Bison.
- Num-samples refers to number of generated samples per query.
- γ optimized for each setting – Tandem can use larger γ more effectively.

Part II: High Recall Top-K Estimation (HiRE)

Matrix-matrix vs matrix-vector multiplications



Takes the same amount of time as



Sparsity and Top-k to the rescue

- The response is sampled from top-k (out of a very large vocab) outputs of Softmax layer.
  - **Can we approximately compute the top-k quickly?**
  - Transfer only those embeddings to VMEM.

- Hidden activations of Feedforward layers have been shown to be sparse. Top-k further amplifies sparsity [1].
  - **Can we approximately estimate the top-k efficiently?**
  - Transfer only the corresponding model parameters to VMEM.

- Each token in attention layer attends to only a few other tokens. [2]
  - **Can we predict which tokens are relevant? Will top-k help here as well?**
  - Transfer only the corresponding keys and values to VMEM.

Key challenges

- Efficient prediction of top-k without doing full computation.
- Efficient gather operation.
  - Multi device sorting and gather
  - Efficiency of gather operations

Related work and key insights

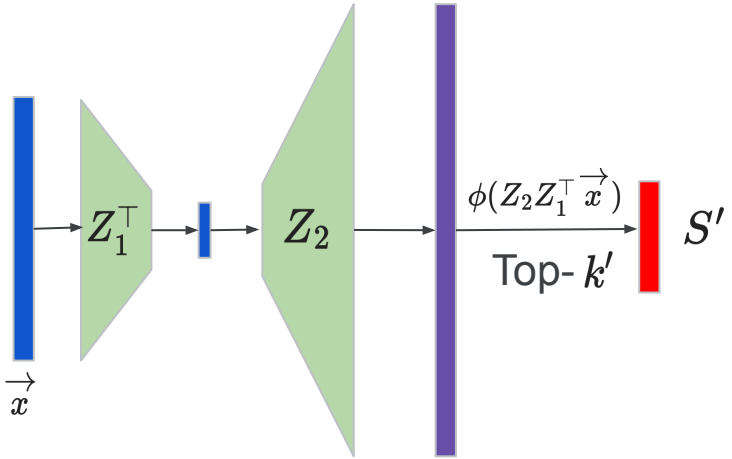
- [1,2,3] train a small network to predict non-zero activations, but suffer accuracy loss due to inexact predictions.

- **Key insights**:
  - Work with top-k trained models instead of relying on natural sparsity.
    - Group top-k to increase memory bandwidth efficiency.
  - High recall estimation by using a larger k' > k.
  - Distributed sorting and gathering of weights.

[1] Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time by Liu et al. ICML 2023
[2] LLM in a flash: Efficient Large Language Model Inference with Limited Memory by Alizadeh et al. 2023
[3] Approximating two-layer feedforward networks for efficient transformers by Csordas et al. 2023

Idea I: High Recall Estimation with Approximate Computation

Step 1

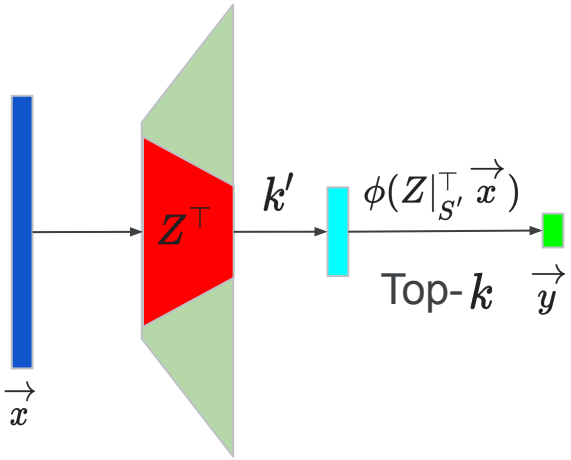


Goal: Compute Top $-k$ $\left(\phi(Z^\top \vec{x})\right)$

Low rank approximation: $Z \approx Z_1 Z_2^\top$

$k' > k$

Step 2



Can also use an aggressively quantized version of Z

Idea II: Distributed Approximate Top-k

- When data/parameters are distributed across multiple machines, perform approximate top-k in a distributed manner.
- Drastically reduces gather and sorting costs, while not losing much recall.

Algorithm 2 Pseudocode for HiRE with DA-TOP-k

input $\overrightarrow{\mathbf{x}}, \mathbf{Z}, \mathbf{Z}_{\text{approx}}, k, k'$ with $\mathbf{Z}$ and $\mathbf{Z}_{\text{approx}}$ distributed across s machines.

1: On machine $i \in [s]$: $S'_i \leftarrow \text{TopInd}_{\frac{k'}{s}}\left(\phi\left({\mathbf{Z}_{\text{approx}}}_i^\top \overrightarrow{\mathbf{x}}\right)\right)$ {Top-$\frac{k'}{s}$ indices of $\phi\left({\mathbf{Z}_{\text{approx}}}_i^\top \overrightarrow{\mathbf{x}}\right)$ on machine i.}

2: $\overrightarrow{\mathbf{y}}_i \leftarrow \text{Top}_{\frac{k}{s}}\left(\mathbf{Z}|_{S'}^\top \overrightarrow{\mathbf{x}}\right)$

3: $y \leftarrow \text{Concat}(y_i : i \in [s])$.

output $\overrightarrow{\mathbf{y}}$

Idea III: Group Top-k

- Efficiency of gather operation improves drastically with group top-k instead of unstructured top-k.
- **Unstructured top-k**: Gather **k** columns of a d X m matrix.
- **Group top-k**: Gather **k/g** slices of a d X g X (m/g) tensor, where g is the group size.

Highlights

A general idea to exploit sparsity. Two variants

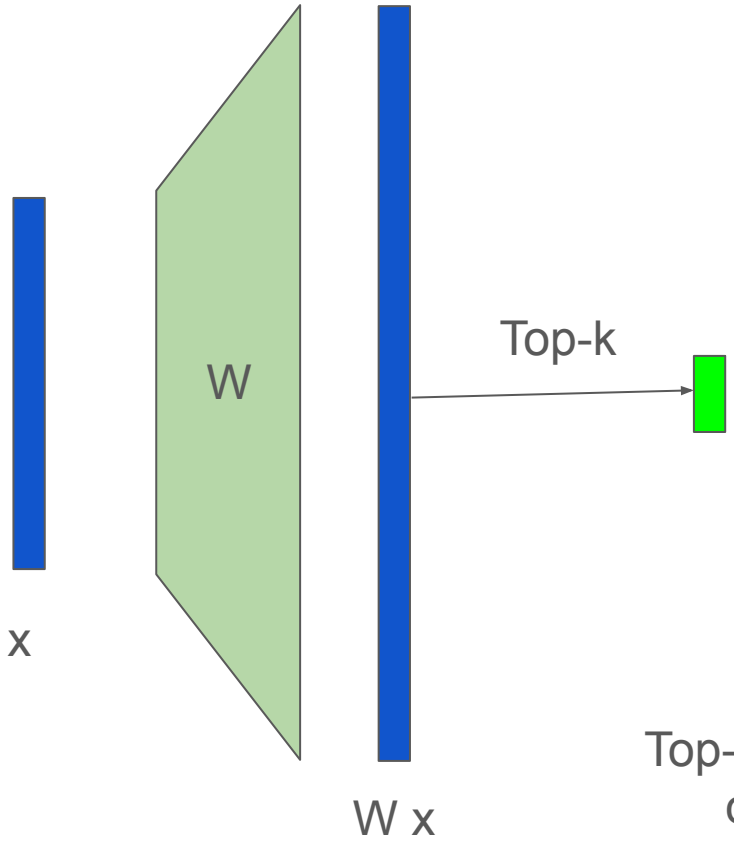
- **HiRE-LR:** Uses low rank approximation. Requires some training.
- **HiRE-Q:** Uses aggressive quantization. Requires no training.

On 1B parameter LLM, HiRE obtains speedups compared to dense model (without top-k) with minimal loss in pre-training and downstream accuracy.

- HiRE-Softmax: 1.22x
- HiRE-FFN + Softmax: 1.47x

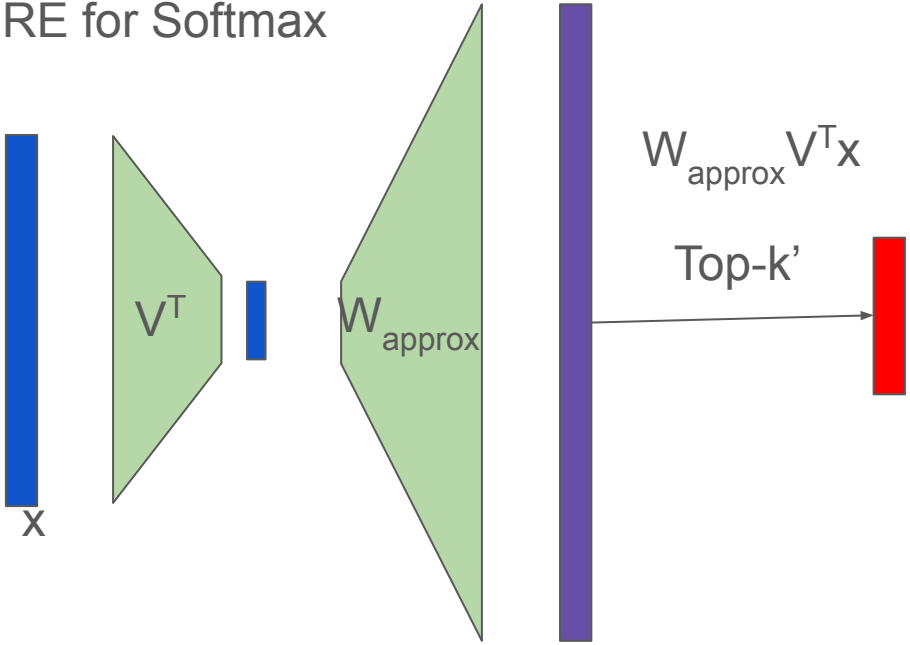
On larger models, much higher scope for latency improvement, but need to train with top-k.

Softmax layer



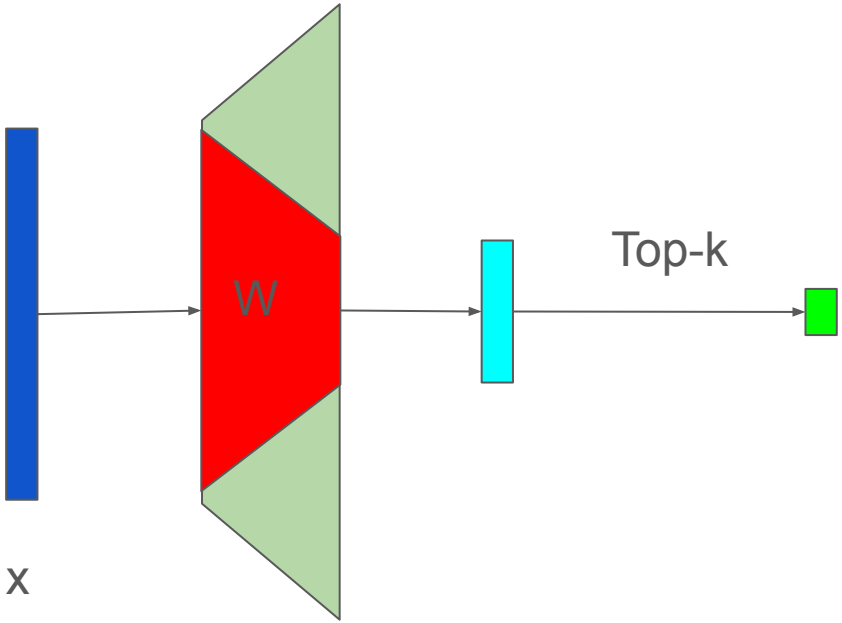
Top-k logits out of large vocabulary

HiRE for Softmax

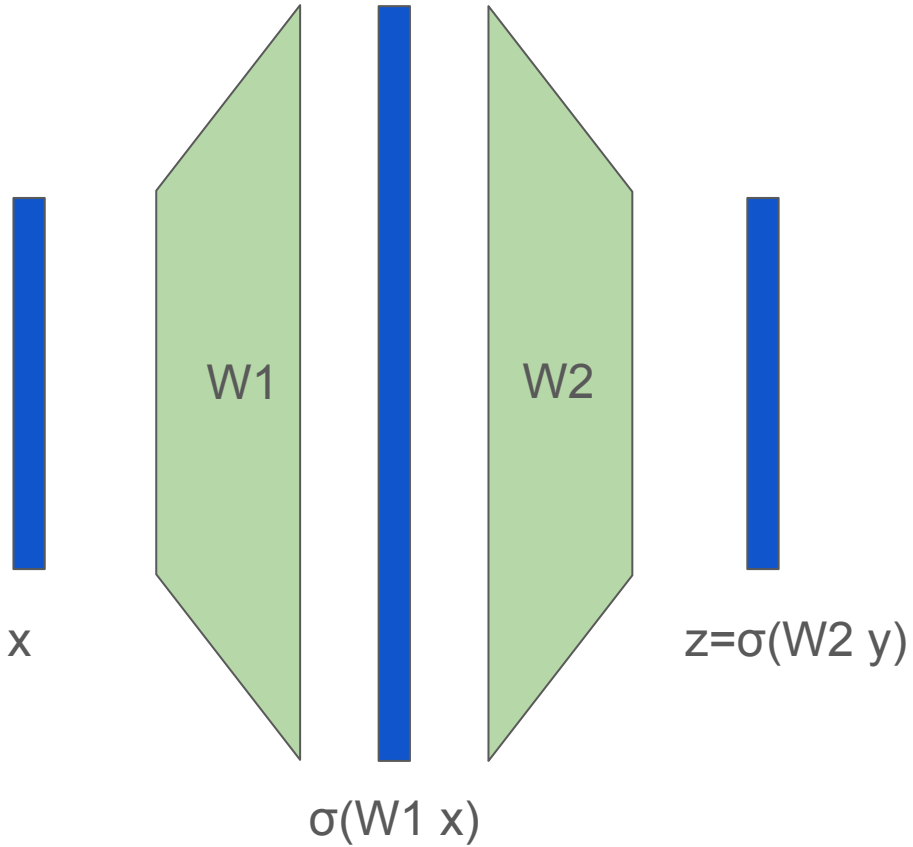


Phase I: Compute approximate logits using low rank approximation of weight matrix.

Phase II: Compute the exact logits **only** for the top-k' approx logits
Requires: gathering only these k' rows of the matrix



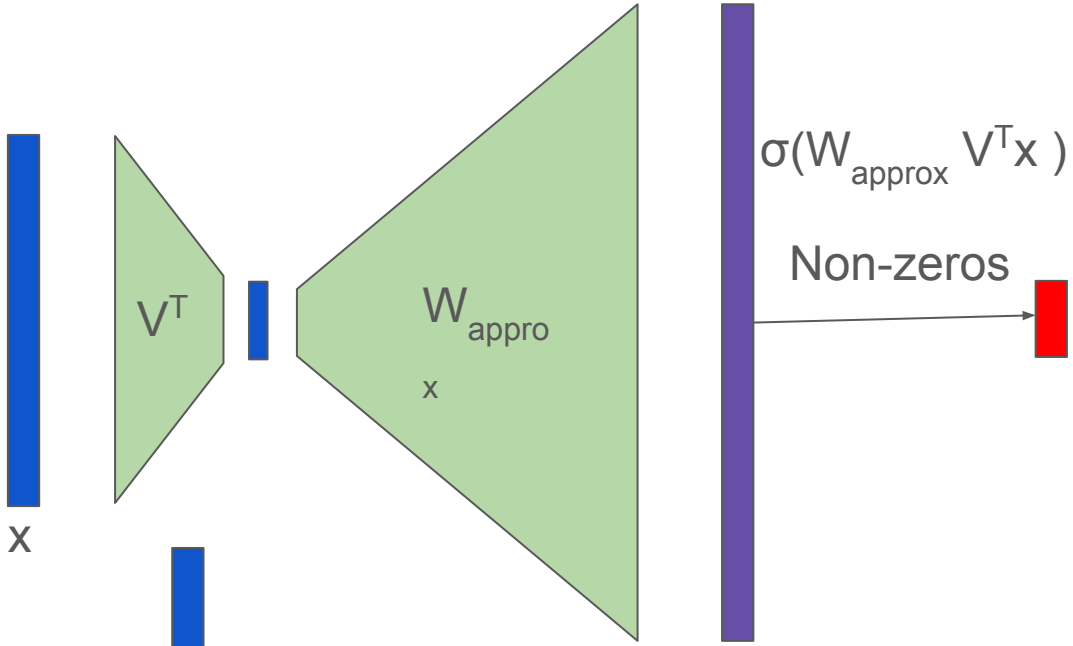
FFN Layer

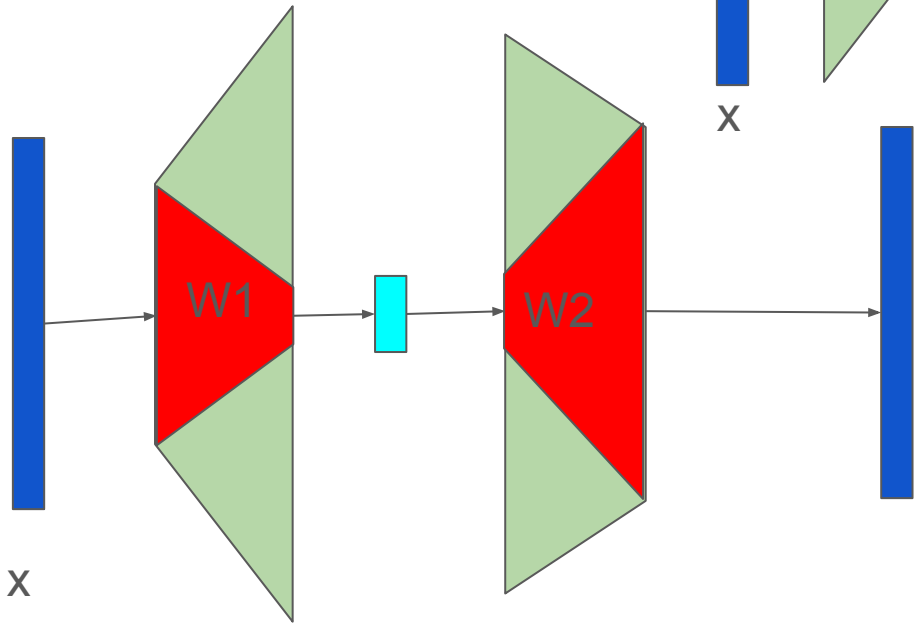


Trained with top-k

HiRE – FFN Layer

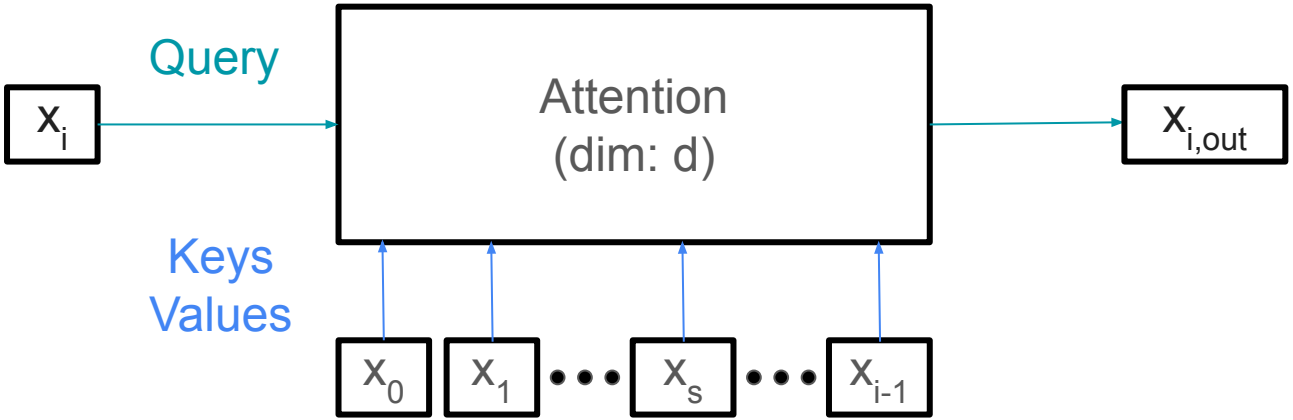
Phase I: Compute the approximate top activations of ffn1 using low-rank approx.



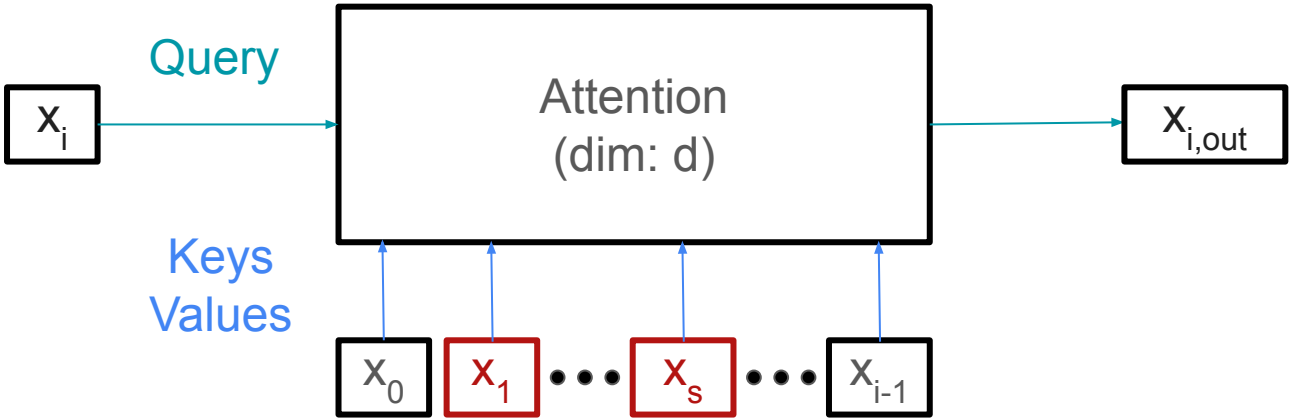


Phase II: Gather only columns of FFN2 from top approx activations and perform exact FFN2 computation on those

Attention Layer



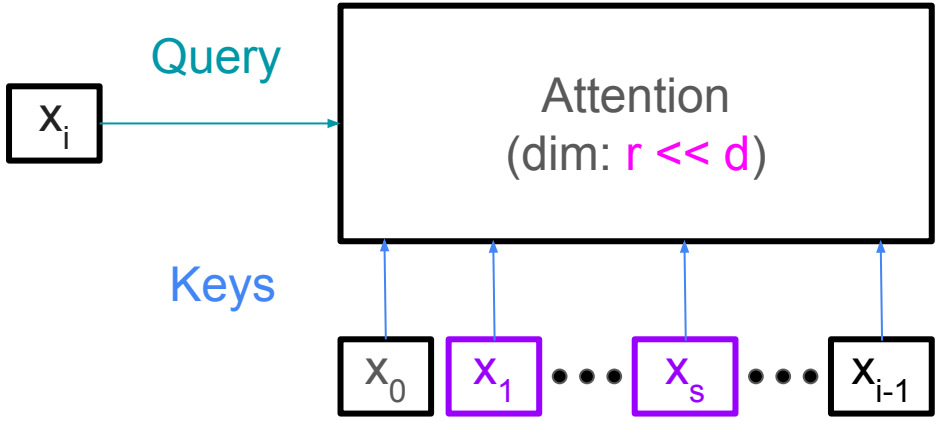
Attention Layer

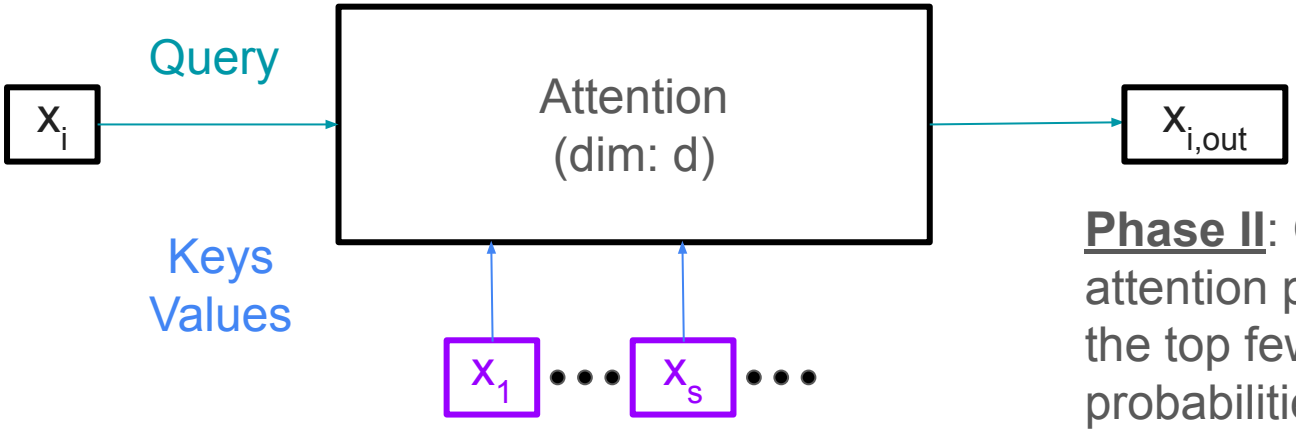


- Each token effectively attends to only a few other tokens [1].
- Earlier works directly use approximate Query Key multiplication, or use a pre-specified sparse token selection model, which usually leads to drop in accuracy.
- In contrast, we would like to use the dense model but identify the relevant tokens.

HiRE – Attention

Phase I: Compute the approximate attention probabilities using lower dimensions.





Phase II: Compute the exact attention probabilities only for the top few approximate probabilities, by gathering keys and values only for those tokens.

Results I

Google Research

- HiRE-Softmax for 1B parameter top-k model

		Baseline	Reduced Dimensions (D=25%, K=384)	Quantization (int4, K=128)	Reduced Dimensions (D=25%, K=384) + Quantization (int4)
Pre-training Performance	Top1 Accuracy	57.15%	57.07%	57.12%	57.07%
	Top32 Intersection	32.0	29.26	31.48	29.03
Downstream Performance	Machine Translation	47.92	47.73	47.9	47.77
	SuperGLUE Benchmark	62.0	61.39	61.56	61.02
	Question Answering	29.65	29.58	29.54	29.58
	Discriminative Tasks	51.69	50.51	50.92	50.40
Per-step Decode Latency		1.0x	1.16x	1.16x	1.22x

Results II

Google Research

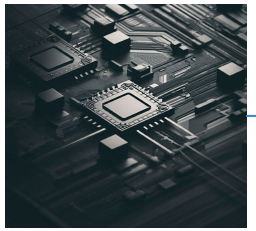
- HiRE FFN + Softmax for 1B parameter top-k model

		Baseline	MLP(Group Sparse) with HiRE-Q (int4)	MLP (Group Sparse) with HiRE-Q (int4) + Softmax with HiRE-Q (int4) & HiRE-LR (384)
Pre-training Performance	Top1 Accuracy	57.15%	57.03%	57.01%
	Perplexity	2.045	2.056	NA
Downstream Performance	Machine Translation	47.92	46.95	46.94
	SuperGLUE Benchmark	62.0	62.49	61.74
	Question Answering	29.65	30.88	30.86
	Discriminative Tasks	51.69	51.14	50.08
Per-step Decode Latency		1.0x	1.16x	1.47x

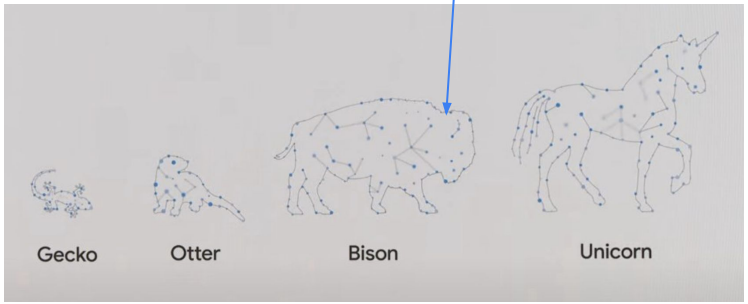
Future work: HiRE for attention layer, dynamic sparsity levels for larger batch sizes.

Part III: Matformers for Elastic Inference

Large Models: Deployment Story



Deployment Constraints
(RAM, Latency, QPS…)

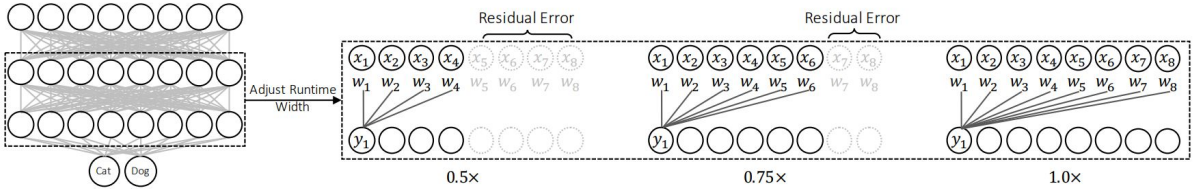


PALM 2

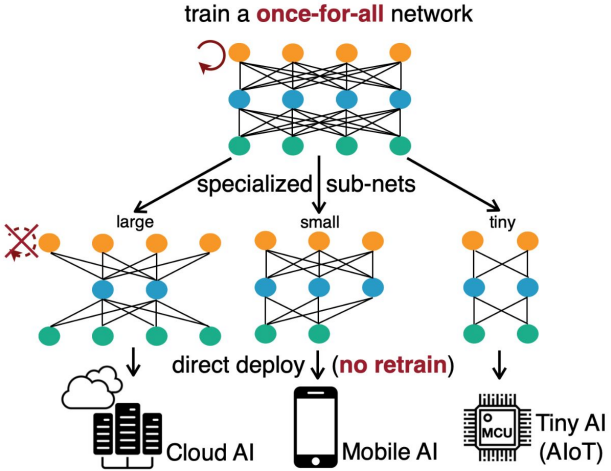
- Typically only a few models to choose from
  - Might have to select say Llama 13B even if capacity for Llama 40B
- Distillation/pruning requires additional training

Goal: *design a "universal" model from which hundreds of accurate models can be extracted*

Existing Solutions towards MatFormer



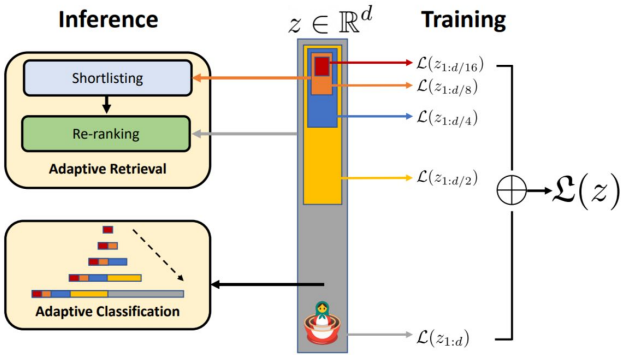
(Universal) Slimmable Networks (Yu & Huang ICLR 2019; ICCV 2019)



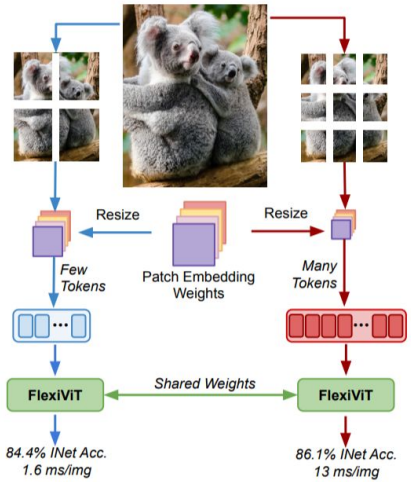
Once-for-All (Cai et al., ICLR 2020)

- Primarily focused on CNNs
- Training routines w/
  - Modifications to batchnorm
  - Sampled submodels
  - Distillation from largest model
- Longer/costlier training routines

Existing Solutions towards MatFormer



Matryoshka Representation Learning (Kusupati et al., NeurIPS 2022)



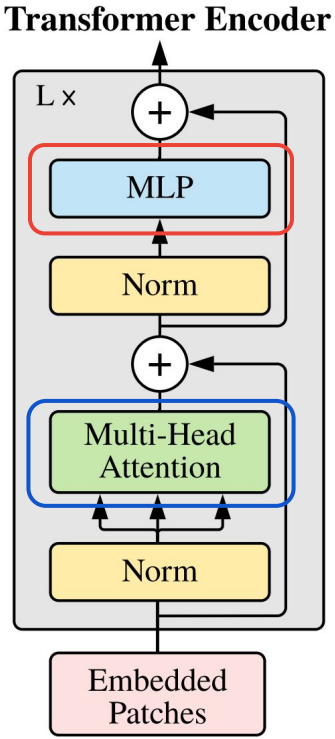
FlexiViT (Beyer et al., CVPR 2023)

Flexibility in output and input space respectively

MatFormer: Nested Substructure

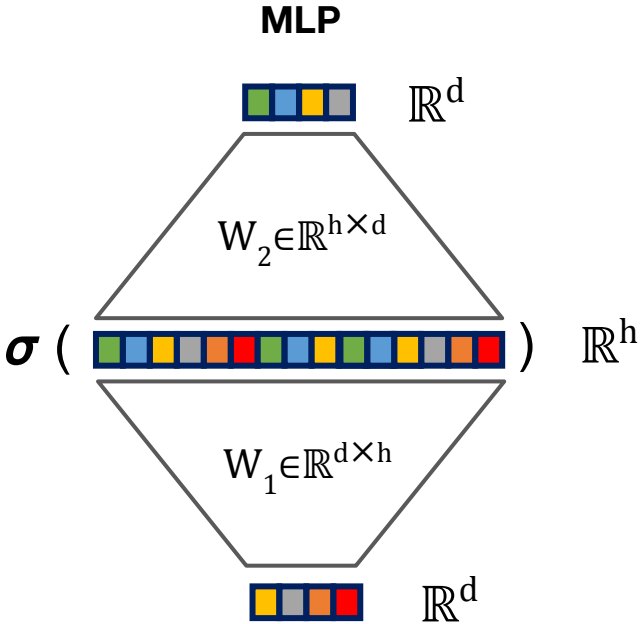
- Works for Transformers – without modifications to fundamental blocks
- Joint training: No subsampling or distillation
- Significantly cheaper training cost that equivalent methods while being more accurate
- 4 granularities sufficient to span a wide range of constraints.

Transformer



MLP: **~80%** of the cost

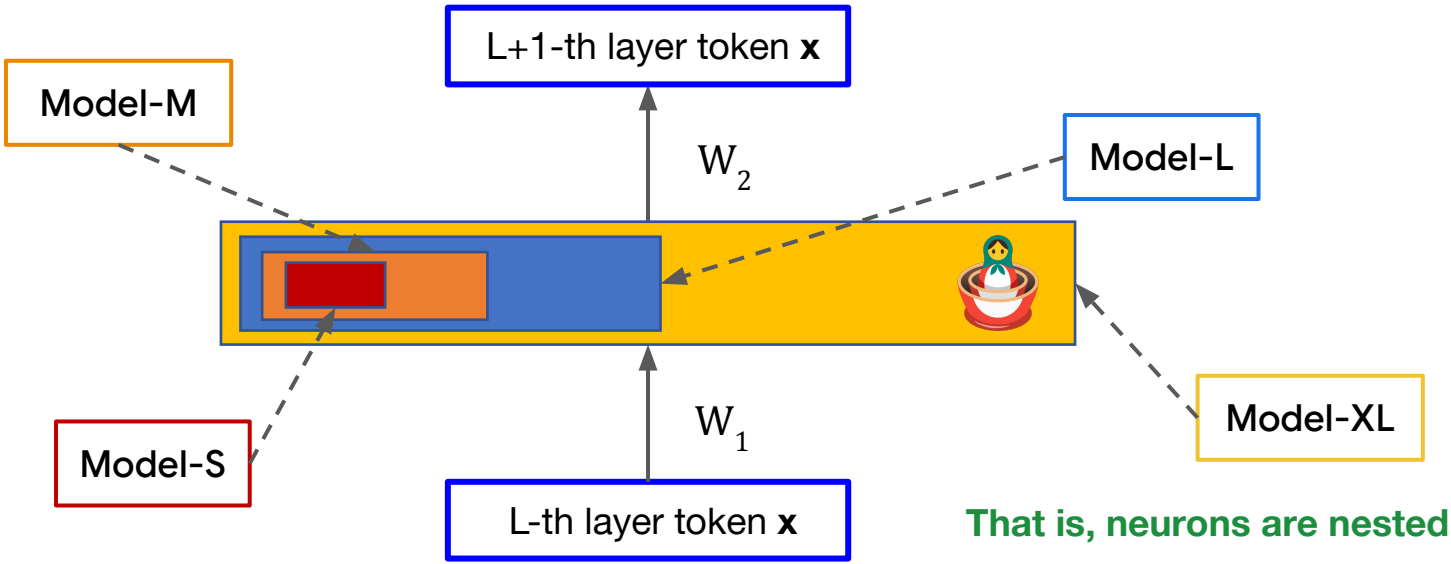
Attention: **~20%** of the cost



h = **4*d** to **12*d** typically

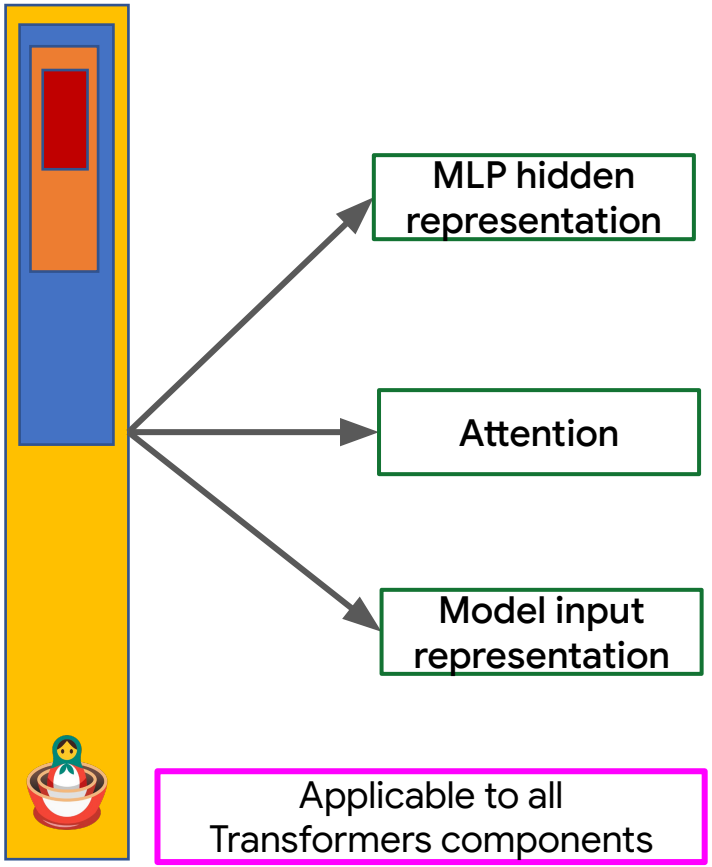
MatFormer: Matryoshka Transformer

- **MatFormer** builds upon MRL
- Apply MRL to MLP layer in each transformer block



That is, neurons are nested in each other.

MatFormer: Generality & Training

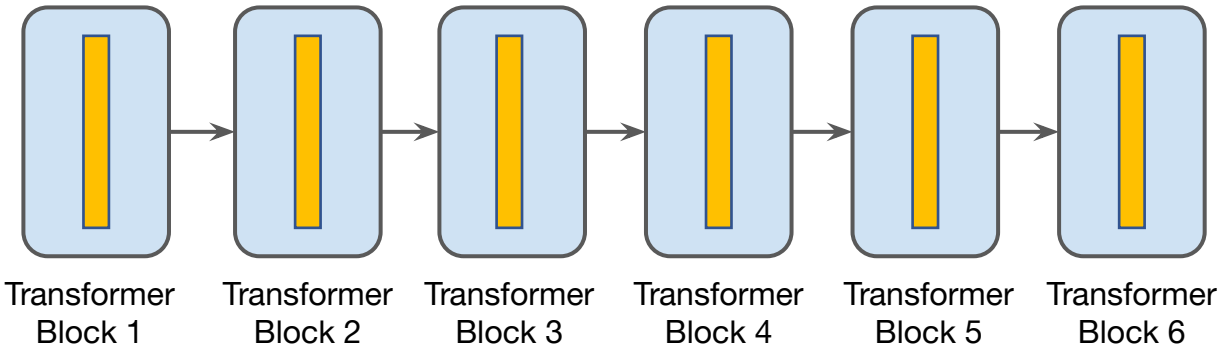


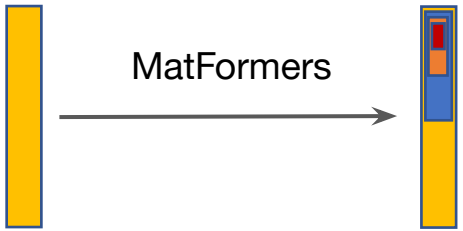
Recipe:

- Pick XL model architecture
- Pick **G** granularities for nesting eg., **G = 4**
- Jointly optimize **G** shared models akin to MRL
- Matformer train cost **<** **total cost of training each granularity from scratch**
- MatFormer can also be induced w/ Fine-tuning

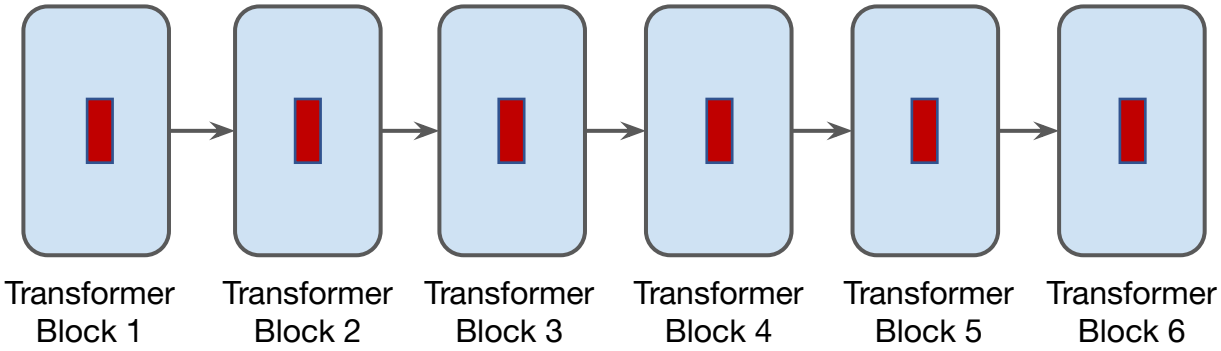
Can generate **1000s** of models not just **4**

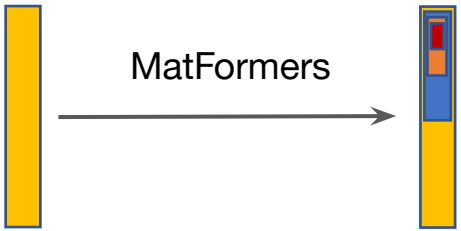
Mix'n'Match & Routing on MatFormer





Mix'n'Match & Routing on MatFormer

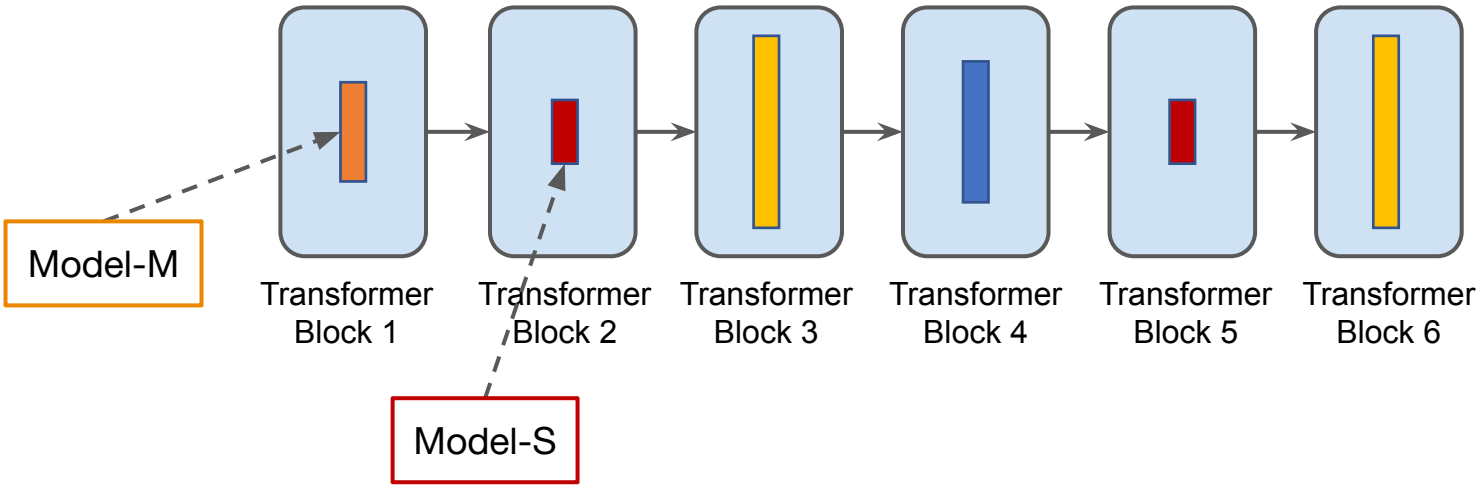




This gives only say 4 models.

So where do we get 1000s of models???

Mix'n'Match & Routing on MatFormer



- **Mix’n’Match:** 100s (combinatorial) of *static (on-demand)* models for all accuracy-compute
- **Routing:** Token based routing akin to MoE to realize *dynamic* computation

MatLM: MatFormers for Language Modeling

- Standard setting from Lamda (Thoppilan et al.)
- **G = 4** granularities – change the MLP hidden dims!
  - **XL** – hidden_dim (hd), **L** – hd/2, **M** – hd/4, **S** – hd/8.
- Nomenclature: MatFormer-XL, MatFormer-L, MatFormer-M, MatFormer-S
  - Independently Trained Models: Baseline-XL, Baseline-L, Baseline-M, Baseline-S
- *7 different "XL" model scales:* from 78M up to 2.6B parameters.
  - 78M, 180M, 310M, 463M, 850M, 1.3B, 2.6B

MatLM: Key Findings

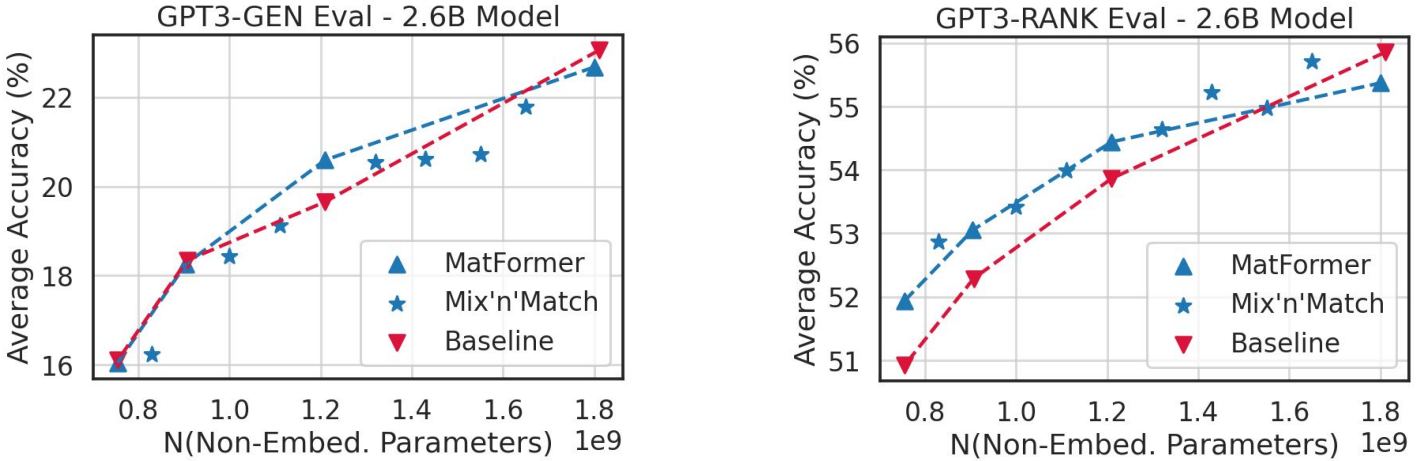
- Little to **no loss in test pplx** and **GPT3 1-shot downstream evals**.
  - For each granularity, i.e., accuracy(MatFormer-Z) ~ accuracy(Baseline-Z)
  - Z $\in$ [XL, L, M, S]

- Able to read models for *free* using Mix'n'Match
  - Mix'n'Match interpolates well between the 4 granularities

- Side effects: consistency with large model, gains over Speculative Decoding

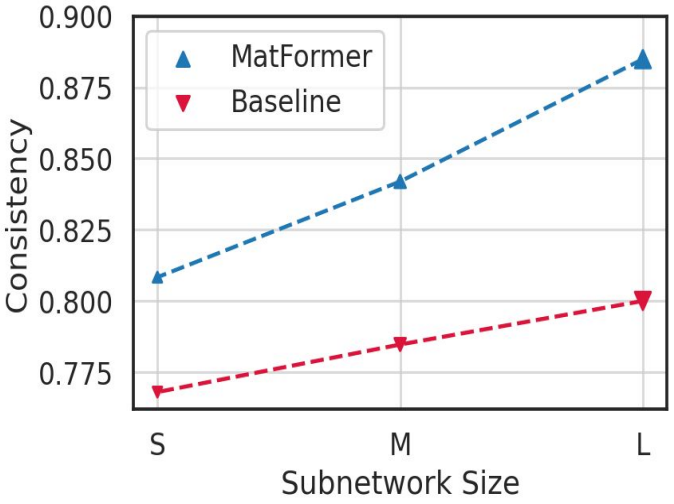
Language Modeling with 2.6B model: Mix’n’Match

1-shot GPT-evals



- Almost matching accuracy for MatFormer–[XL, L, M, S] models against Baselines
- We get all the intermediate models denotes by ★ for “free”
  - No extra training!
  - For GPT-3 Rank: ★ models almost lie on a line interpolating trained MatFormer models (XL, L, M, S)

Language Modeling: Consistency for 2.6B XL model



***Consistency**: accuracy of smaller models (S, M, L) when output of XL model is the ground truth*

Why care about consistency?

Techniques like Speculative Decoding becomes more efficient with more consistent models

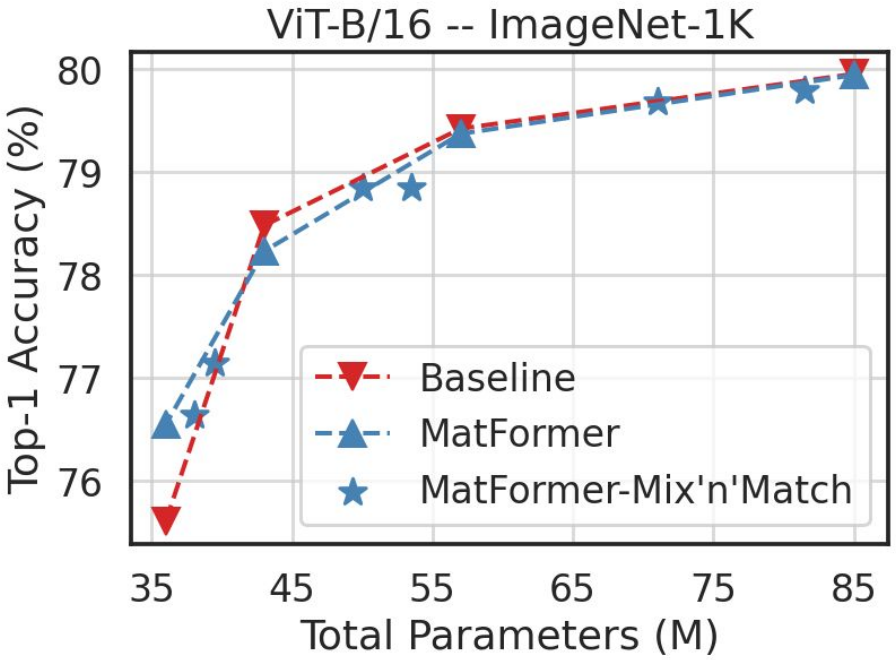
Speculative Decoding	LAMBADA	TriviaQA
Baseline	1.10×	1.08×
MatLM	1.14×	1.11×
+ shared attention cache	1.16×	1.14×

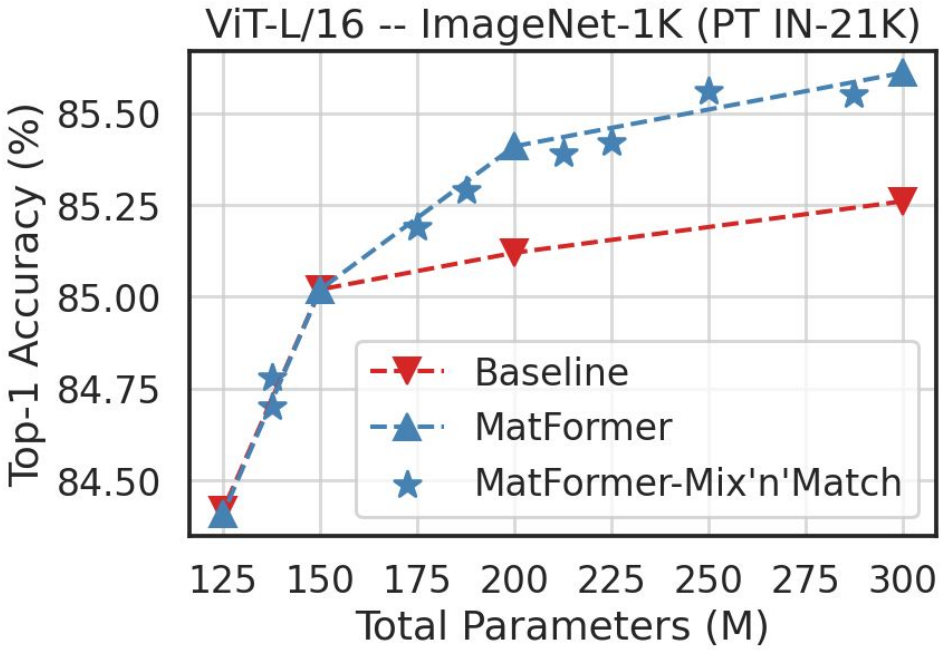
MatFormer subnetworks are significantly more consistent with the full model compared to vanilla baselines.

MatViT: MatFormer + ViT

- Generalized formulation translating to ViT
- Works for across model sizes for both pre-training and fine-tuning
- Enables accurate adaptive encoders for classification
  - Spans all of the space with Mix'n'Match (and potentially routing)
- Enables accurate adaptive query encoders for retrieval
  - Use the largest model for Index building
  - Leverage smaller query encoders during inference based on the constraints
  - This requires aligned training/distillation for baseline models to work

MatViT: Classification





All the ★ are for “free” during inference – *they were never optimized for.*

Conclusions and Future Directions

- Inference latency is a big bottleneck with very large room for improvement.
- Two approaches:
  - HiRE for efficient top-k
    - 1.47x speedup over dense 1B model with minimal loss in accuracy.
    - Current work for softmax and FFN → Extend to attention.
    - Exploiting top-k overlap across tokens in a batch
  - Tandem transformers to mitigate autoregressive component
    - 1.11-1.17x speedup over distilled models in SPEED
    - Other variants of tandem e.g., using LLM for plan generation and SLM for autoregressive token generation using plan.
    - Tandem as an alternative to LoRA for finetuning.