

Scaling Up Large Language Models Serving Systems For Semantic Job Search

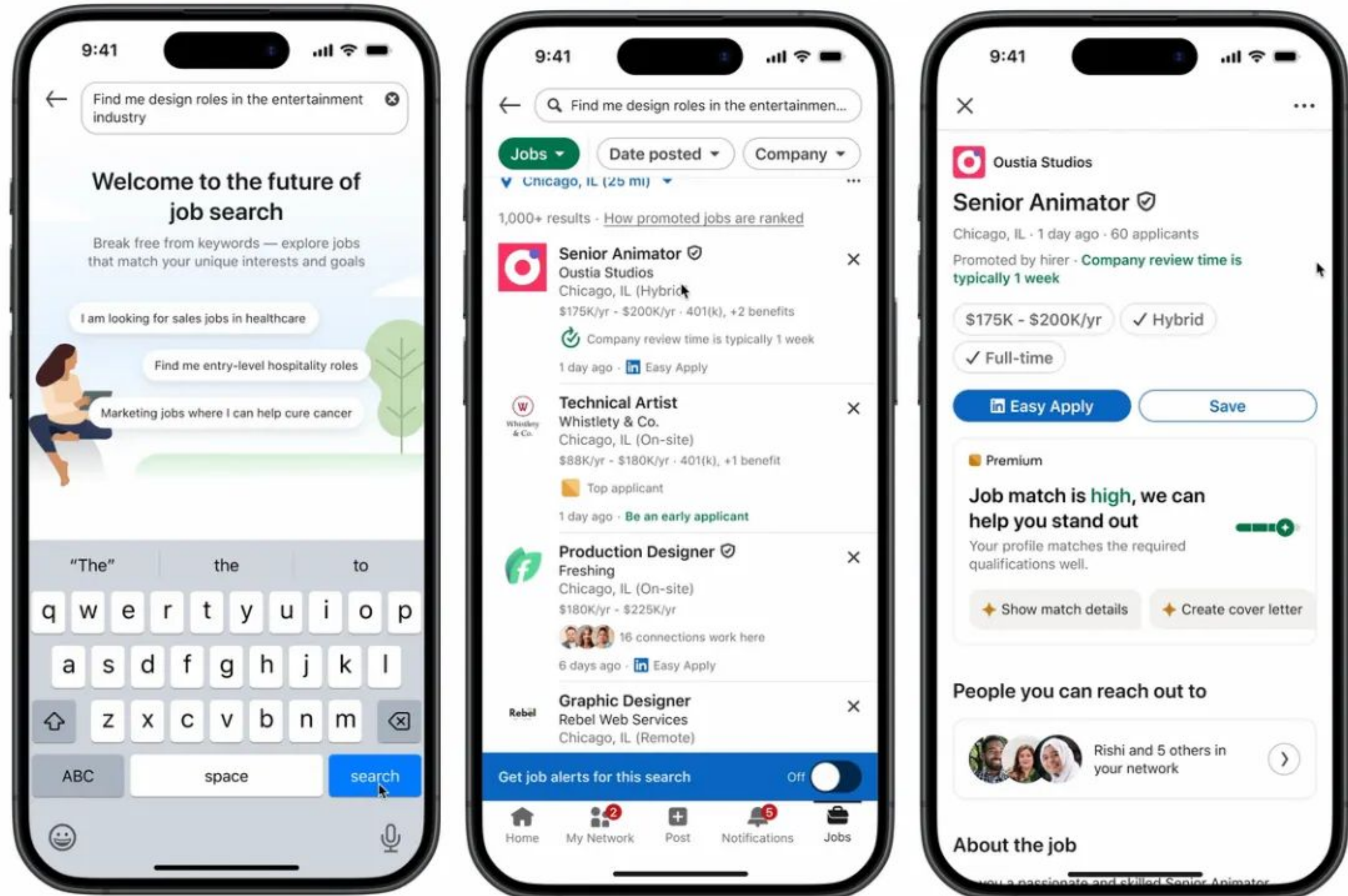
Presenters: Zhipeng Wang, Sundara Raman Ramachandran

Core Team: Kayhan Behdin, Qingquan Song, Zhipeng Wang, Sundara Raman Ramachandran, Chanh Nguyen, Jian Sheng, Qing Lan, Fedor Borisyyuk, Sriram Vasudevan et al.

May 2026

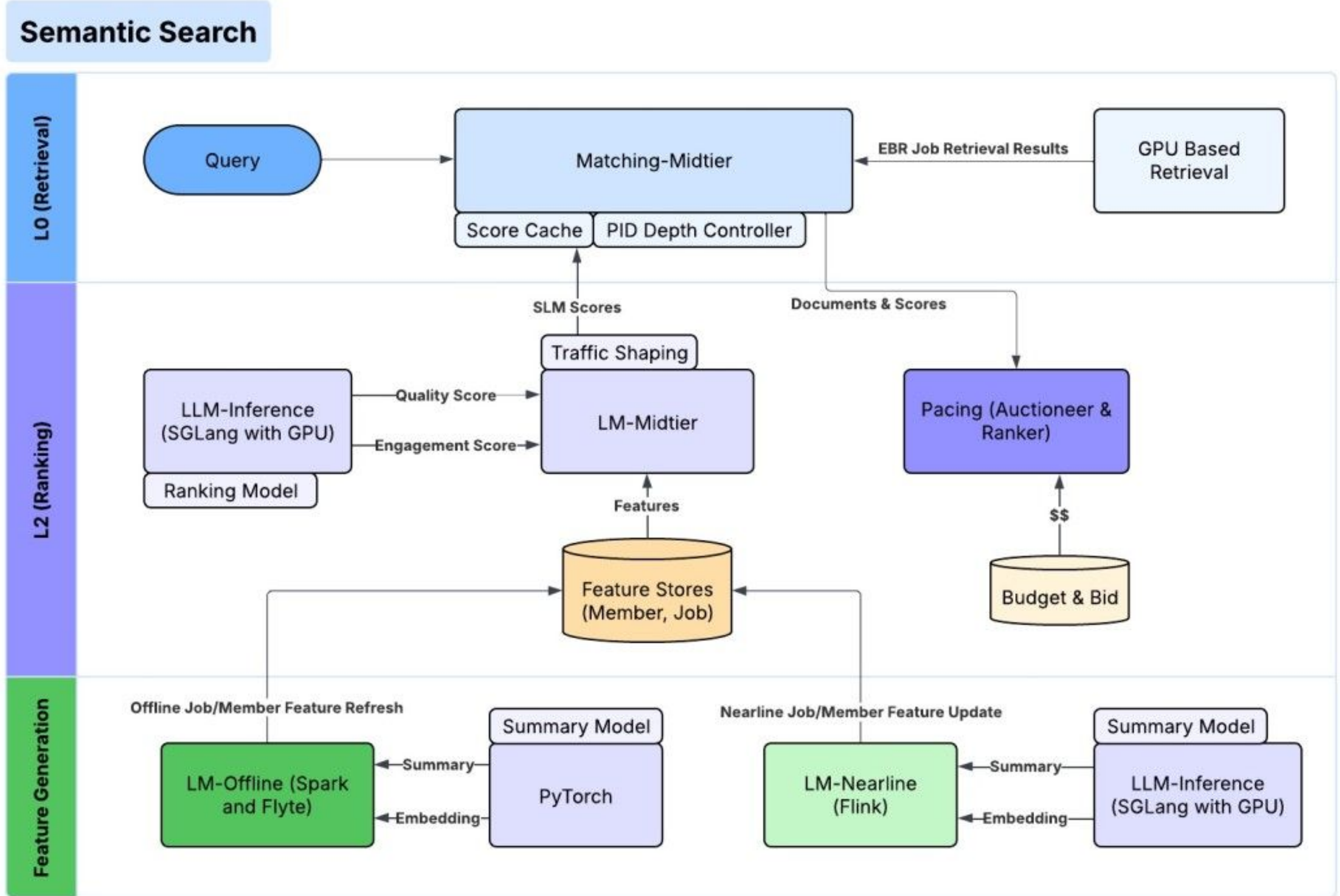
Linked 

LLM-powered Semantic Job Search Features on LinkedIn



Semantic Job Search at LinkedIn

Search Query (natural language form: e.g. "find me the jobs of Machine Learning Engineers in Seattle area")
 +
 Item Description (e.g. Job Description)
 ↓ **LLM Ranker**
 Single Label Token (Yes/No, e.g. Apply, Click etc.)
 ↓
 Probabilities (e.g. p_{apply} , p_{click})

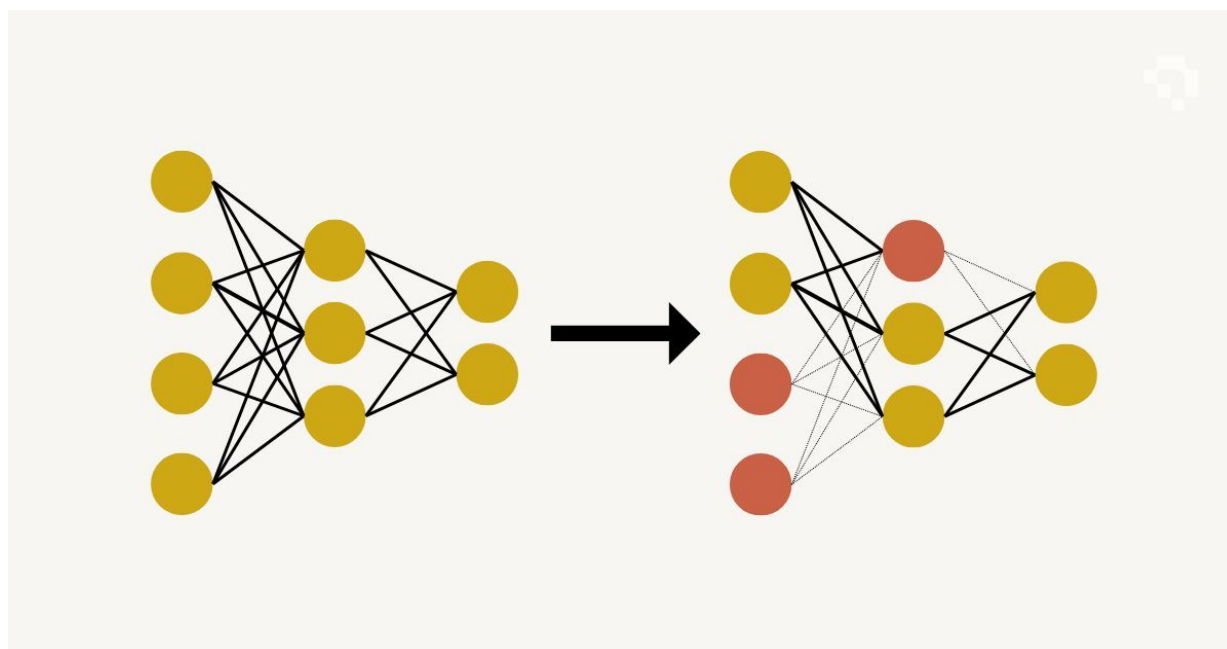


Optimizing LLM Serving Stack for Semantic Job Search

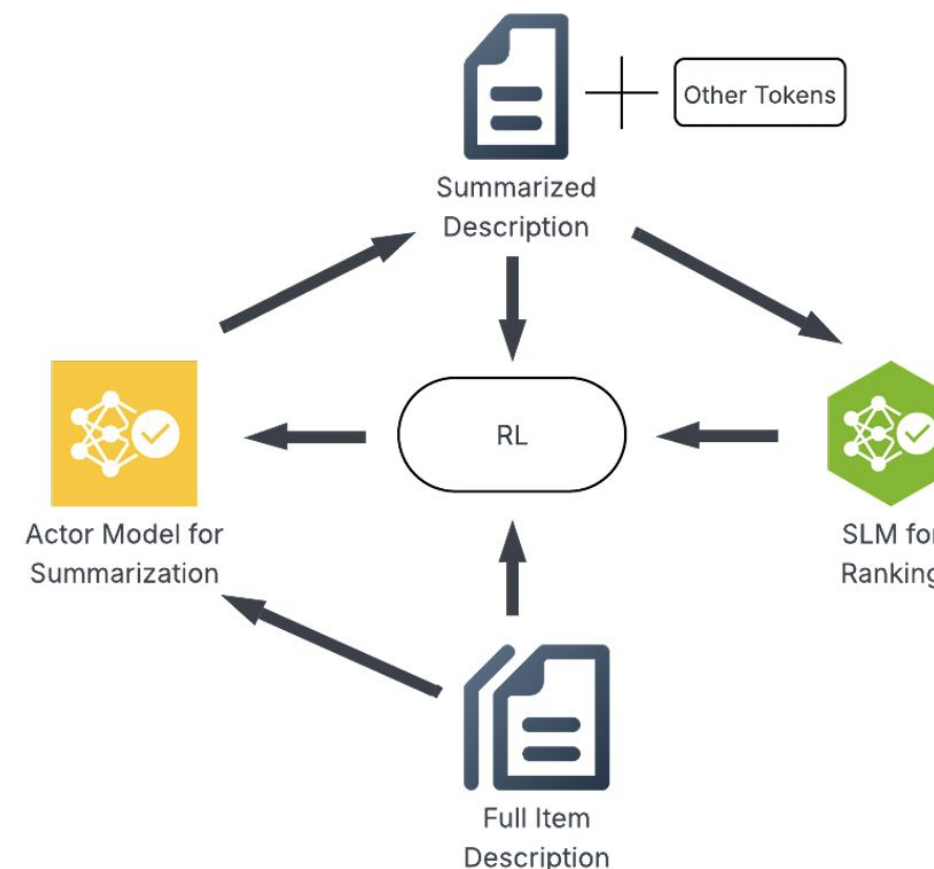
Need to serve millions of requests per second, throughput is definitely a bottleneck

*We achieved **10x throughput improvements** through multi-layer Optimizations and deployed the model in production*

Model Compression via Pruning



Item Description Summarization through RL training



Serving Infra Optimization

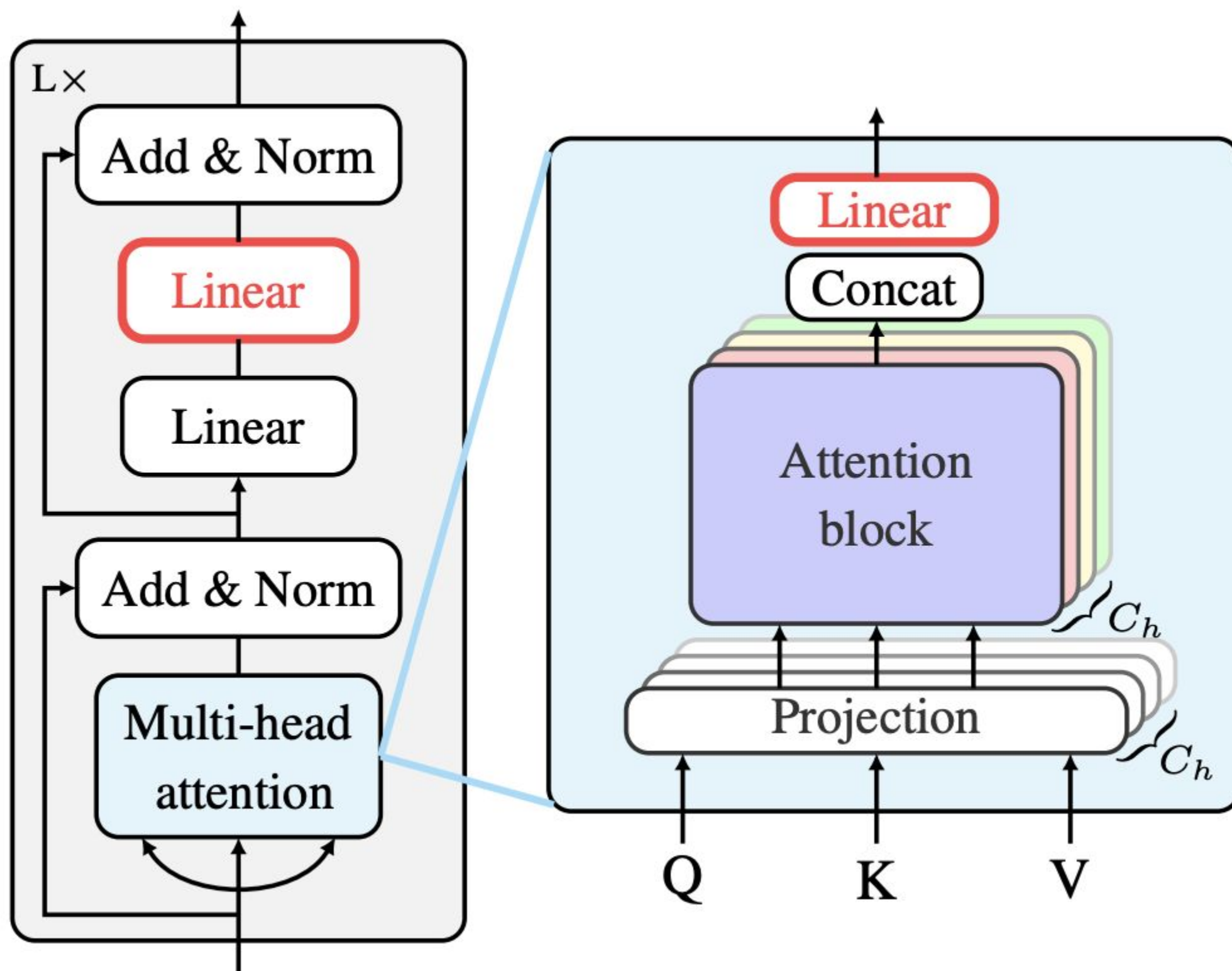


Model Compression via Pruning

- Goal: Serving the Semantic Job Search
- Scoring relevance of (query, item) pairs
- QPS = 3.15M items/sec, latency ~ 500ms
- We performed model pruning using one-shot structured pruning OSSCAR (Meng et al., ICML 2024) to reduce the model size:
 - Pruning hidden neurons in feed-forward MLP
 - Removing model layers and transformer blocks
- Increased throughput by 30%, less than 1% accuracy loss

MODEL SIZE	NDCG@10 CHANGE
600M	-
375M (50% MLP + 8 LAYERS)	-0.0079
350M (50% MLP + 10 LAYERS)	-0.0074
330M (50% MLP + 12 LAYERS)	-0.0080

OSSCAR Pruning (Meng et al., ICML 2024)



MODEL	NDCG@10 CHANGE
UNPRUNED	-
50% MLP PRUNING	-0.0095
50% MLP PRUNING + SFT	-0.0046

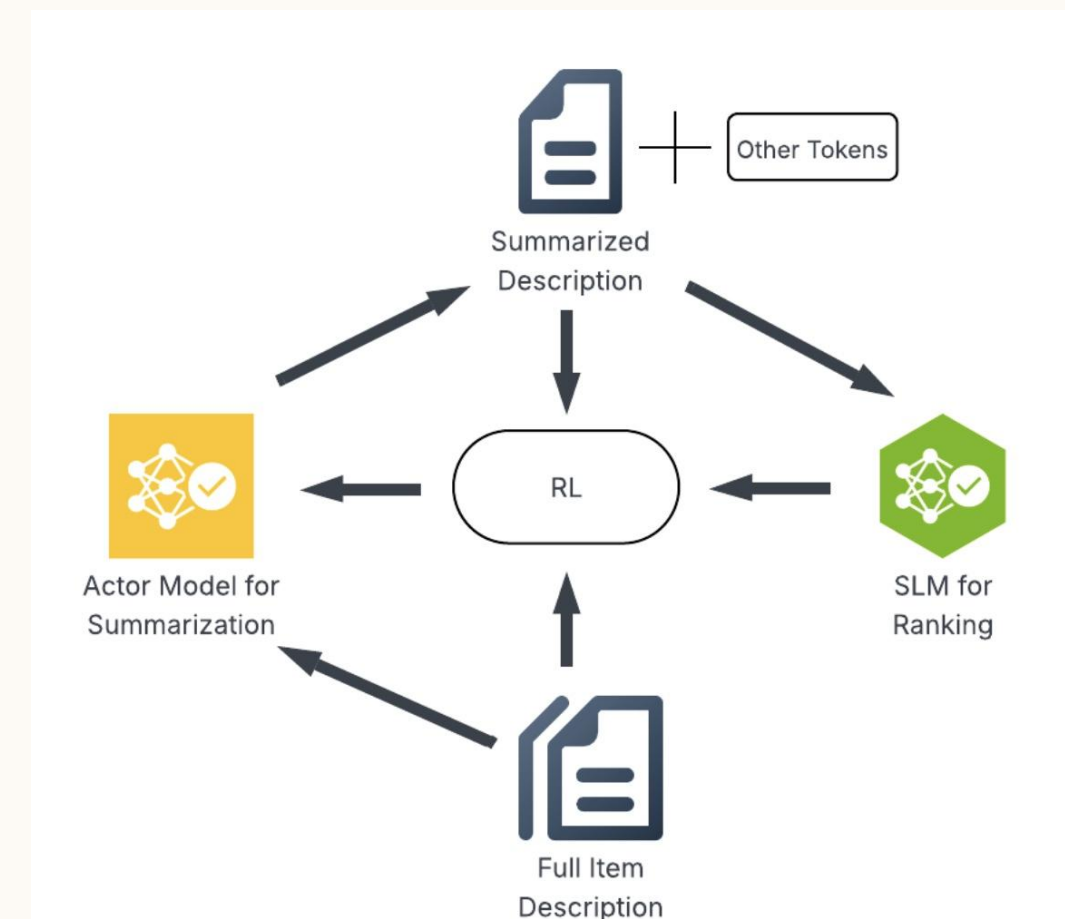
LAYER TO REMOVE	NDCG@10 CHANGE
NO LAYER REMOVED	-
FIRST LAYER	-0.3356
SECOND LAYER	-0.0296
THIRD LAYER	-0.0133
TENTH LAYER	-0.0166
TWO BEFORE LAST LAYER	-0.0001
ONE BEFORE LAST LAYER	-0.0004
LAST LAYER	-0.0009

Item Description Summarization

- Job description can take >2k tokens
 - Reduced throughput
- Removing job description leads to the ranking quality drop(e.g. NDCG drop)
- We post-trained an Actor Model for Summarization via RL training:
 - Reduce the item description length to ~120 tokens
 - Less than 2% *NDCG@10* loss

$$\text{reward} = -KL(p_{\text{sum}} || p_{\text{raw}}) - w \left(\frac{\text{len}_{\text{sum}}}{\text{len}_{\text{raw}}} \right)^2.$$

MODEL SIZE	SFT ON SUMMARIZED	EVALUATION ON SUMMARIZED	NDCG@10
600M	✗	✗	0.8950
375M	✗	✓	0.8786
375M	✓	✓	0.8788



$$\text{reward} = -KL(p_{\text{sum}} \| p_{\text{raw}}) - w \left(\frac{\text{len}_{\text{sum}}}{\text{len}_{\text{raw}}} \right)^2.$$

Two types of length penalties

$$P_1(L_o, L_c) = \begin{cases} 0, & \text{if } L_o < m \\ & \text{or } r \leq \tau, \\ -w \left(\frac{r - \tau}{1 - \tau} \right)^2, & \text{otherwise,} \end{cases} \quad P_2(L_o, L_c) = -wr^2$$

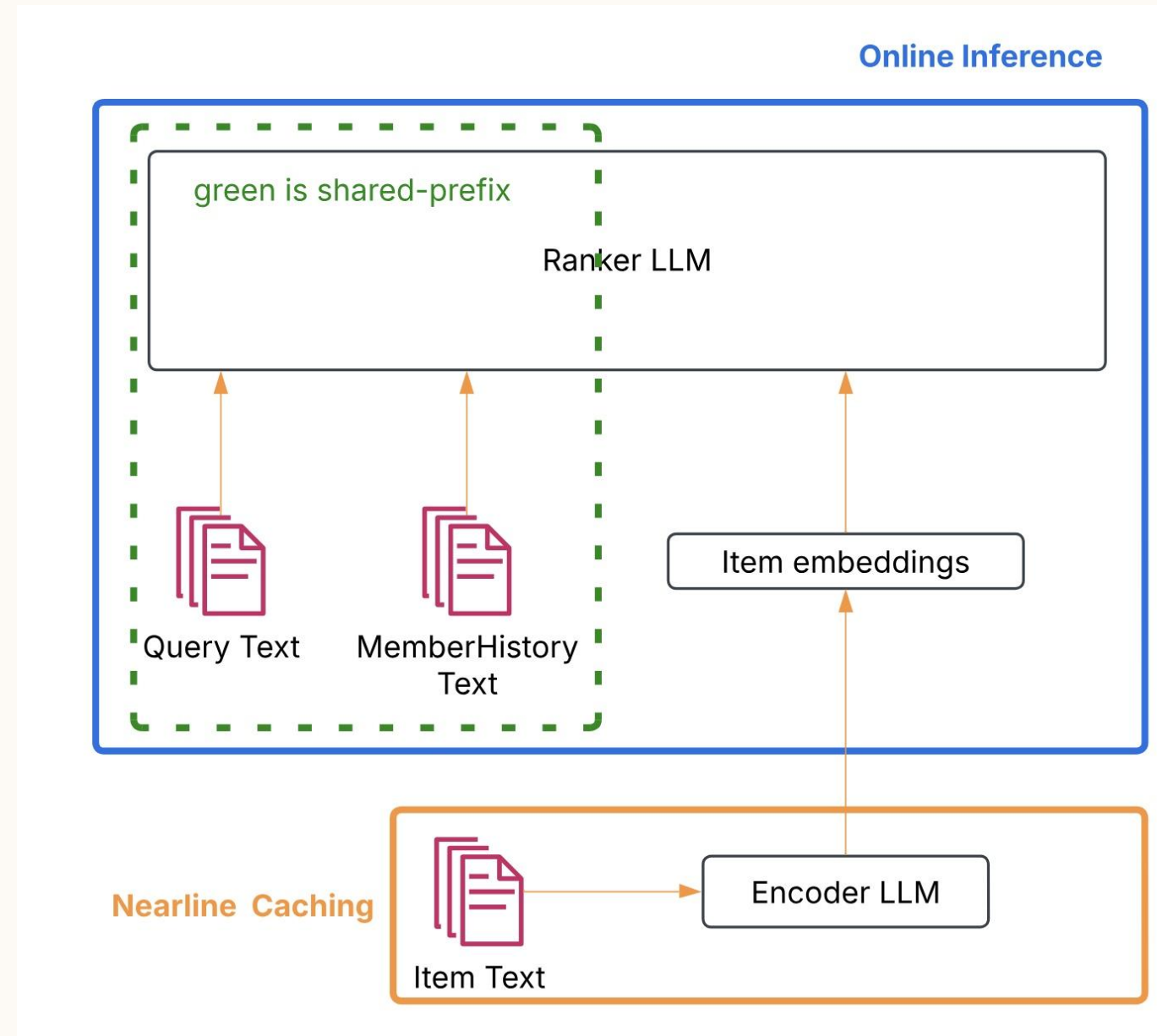
The RL training pipeline is built on top of verl and we explored GRPO, DR. GRPO and GSPO etc.

ALGORITHM	LENGTH PENALTY	PENALTY WEIGHT	NDCG@10	COMPRESSION
“SUMMARIZE” PROMPT	-	-	-2.5%	-52%
GRPO	P_1	$w = 0.0$	-1.8%	-52%
DR. GRPO	P_1	$w = 0.0$	-1.0%	-66%
DR. GRPO	P_1	$w = 0.05$	-1.1%	-69%
DR. GRPO	P_1	$w = 0.1$	-1.6%	-84%
DR. GRPO	P_1	$w = 1.0$	-2.0%	-92%
DR. GRPO	P_1	$w = 10.0$	-2.4%	-94%
DR. GRPO	P_2	$w = 0.1$	-2.0%	-89%
DR. GRPO	P_2	$w = 0.5$	-2.2%	-94%
DR. GRPO	P_2	$w = 1.0$	-2.6%	-95%
DR. GRPO	P_2	$w = 10.0$	-3.9%	-98%
GSPO	P_2	$w = 0.3$	-1.7%	-91%
GSPO	P_2	$w = 0.4$	-2%	-93%
GSPO	P_2	$w = 1.0$	-2.2%	-95%

(Extension) Embedding compression

- Idea: Compress each item text into embedding tokens
- Keep the query as text
- Reducing context from >2000 tokens to ~50
- Up to 50% further throughput increase

Setup	V8 NDCG@10
All text Prod baseine (Sept, pruned and summarized)	0.921
Text + 1 embedding token	0.922



- We open-sourced our foundation model optimization algorithm library(**fmchisel**), which contains components such as Model Pruning, Knowledge Distillation, Quantization, Training Acceleration and Prototyping etc.)
- It's plug-and-play and can be applied to a wide range of use cases.

fmchisel Library

<https://github.com/linkedin/fmchisel>



📖 README 📄 BSD-2-Clause license

fmchisel – Efficient Foundation Model Algorithms

State-of-the-art compression & distillation recipes for Large Language Models

🌟 Overview

fmchisel (*Foundation Model Chisel*) is an **open-source research library** that makes it simple to:

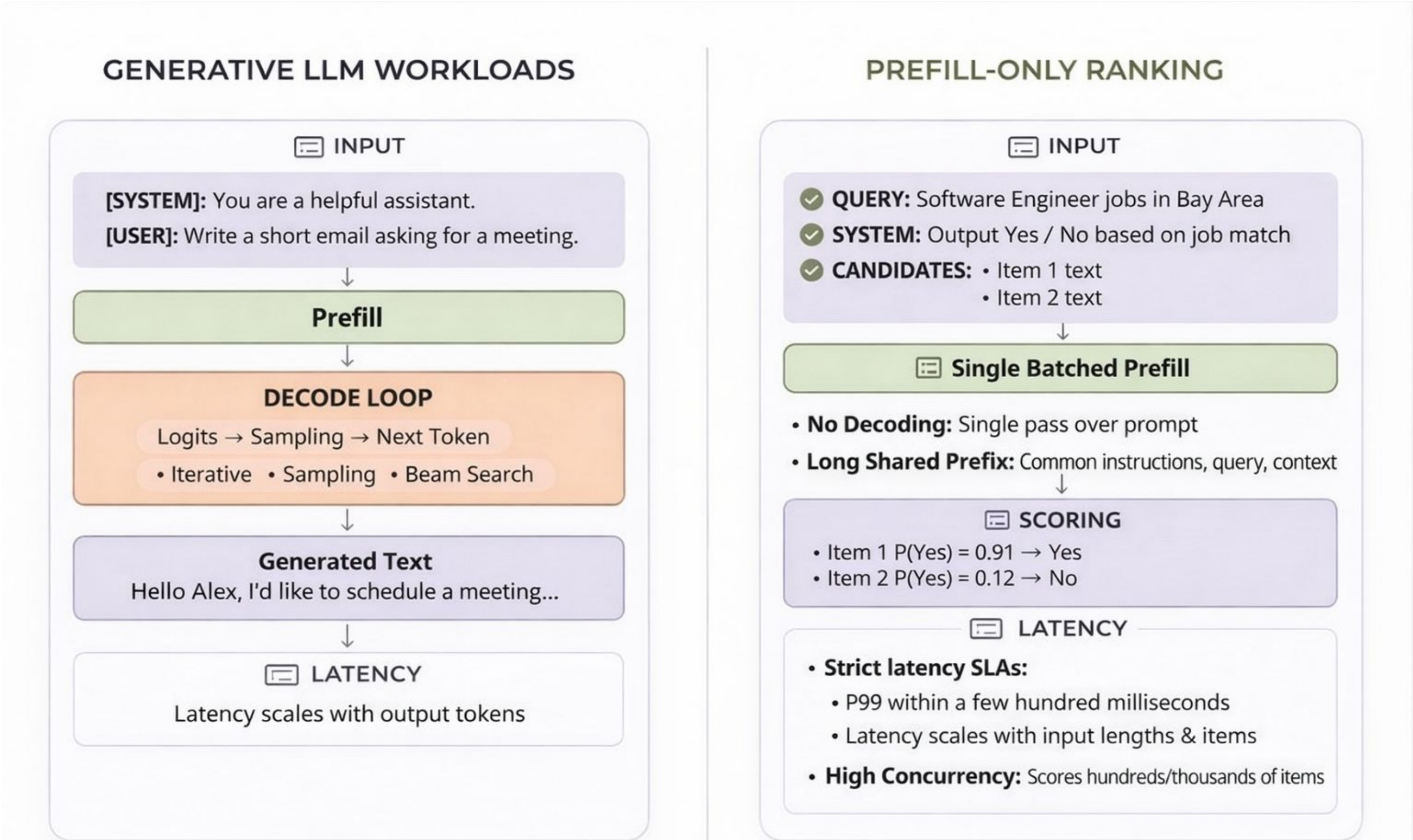
- **Compress** LLMs with cutting-edge pruning and quantization techniques.
- **Distill** knowledge from larger models to smaller ones.
- **Accelerate** inference on consumer hardware by combining sparse + low-bit weight formats.
- **Train** efficiently with advanced optimizers such as schedule-free **AdamW**.
- **Prototype** new compression ideas rapidly.

fmchisel is built on **PyTorch** and integrates seamlessly with 📖 🤗 **Transformers**.

Related Publications

- *Scaling Down, Serving Fast: Compressing and Deploying Efficient LLMs for Recommendation Systems (**EMNLP** 2025)*
- *Reasoning Models Can be Accurately Pruned Via Chain-of-Thought Reconstruction (**ICLR** 2026)*
- *LLM Query Scheduling with Prefix Reuse and Latency Constraints (**Neurips** 2025)*
- *Distilling the Essence: Efficient Reasoning Distillation via Sequence Truncation (**ACL** 2026)*
- *Scaling Up Efficient Small Language Models Serving and Deployment for Semantic Job Search (**MLSys** 2026)*
- *OTPrune: Distribution-aligned Visual Token Pruning via Optimal Transport (**CVPR**, 2026)*
- *EVTP-IVS: Effective Visual Token Pruning For Unifying Instruction Visual Segmentation In Multi-Modal Large Language Models (**WACV** 2026)*

Serving Infra Optimization: Why Prefill-Only Ranking is Different



Reusing Query Work

In-batch Prefix Caching

⚙️ How It Works

- Compute prefix KV from first prompt
- Reuse prefix KV for remaining prompts in same batch
- For suffix tokens, merge attention:
 - Prefix attention (attend to shared KV)
 - Suffix attention (causal within item)
 - Combine via log-sum-exp

- ✓ Numerical correctness preserved
- ✓ No change to model semantics
- ✓ Single forward pass

A) In-Batch Prefix Caching (Token 8)

1 Two seqs in a batch (Query as shared prefix [1, 2])

Seq A: [1, 2, 3, 4, 5]

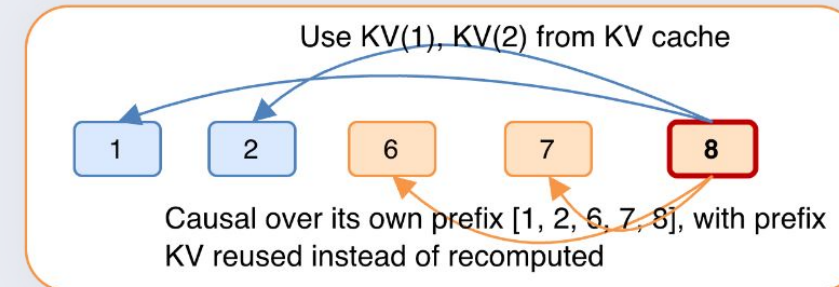


Seq B: [1, 2, 6, 7, 8]

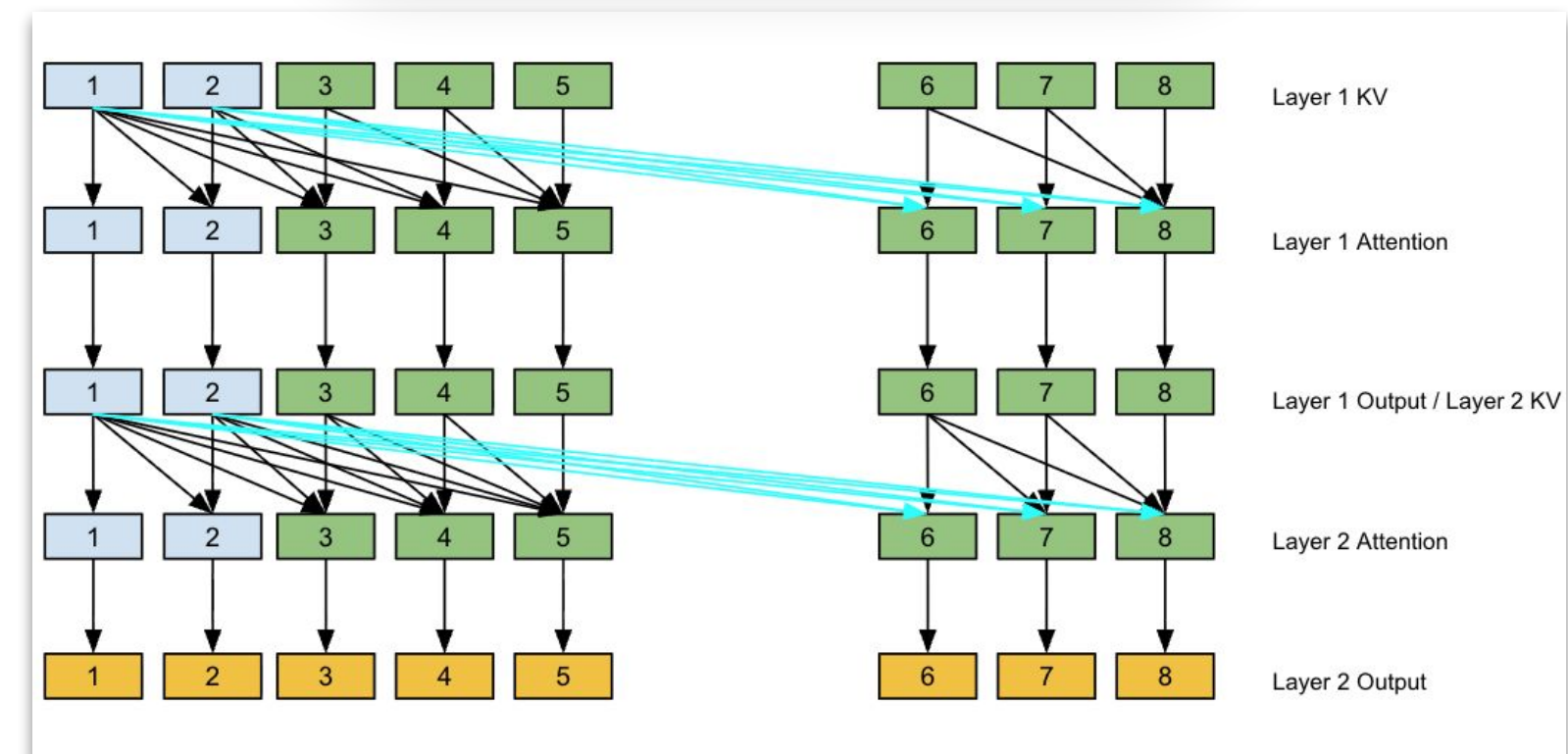


Shared KV cache for prefix [1, 2] (compute once, reuse)

2 Attention for Item token 8: prefix + its own suffix



Result: Same attention scores as naive computation, but prefix work for [1, 2] is shared across seqs.



Reusing Query Work

Multi-Item Scoring

⚙️ How It Works

- Concatenate all items into one sequence

`<Query><DELIM><Item 1><DELIM><Item 2>...<DELIM><Item N><DELIM>`

- Apply item-aware attention mask
 - Shared prefix attends normally
 - Items cannot attend to each other
- Extract scores at item boundaries

Callouts

- ✓ Single forward pass
- ✓ Prefix reused naturally via concatenation
- ✓ No cross-item attention

B) Multi-Item Scoring (Token 8)

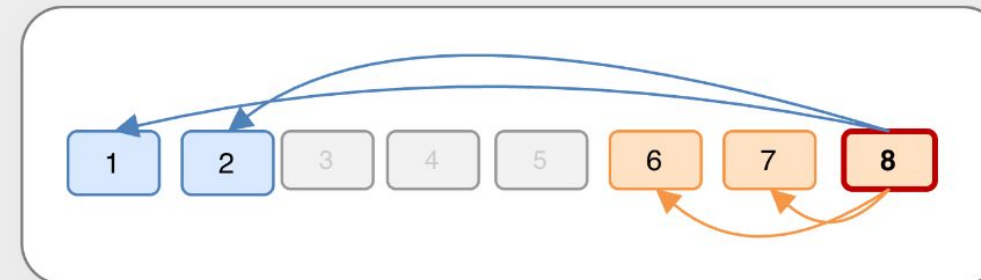
- 1 Single concatenated sequence: Query + Item 1 + Item 2



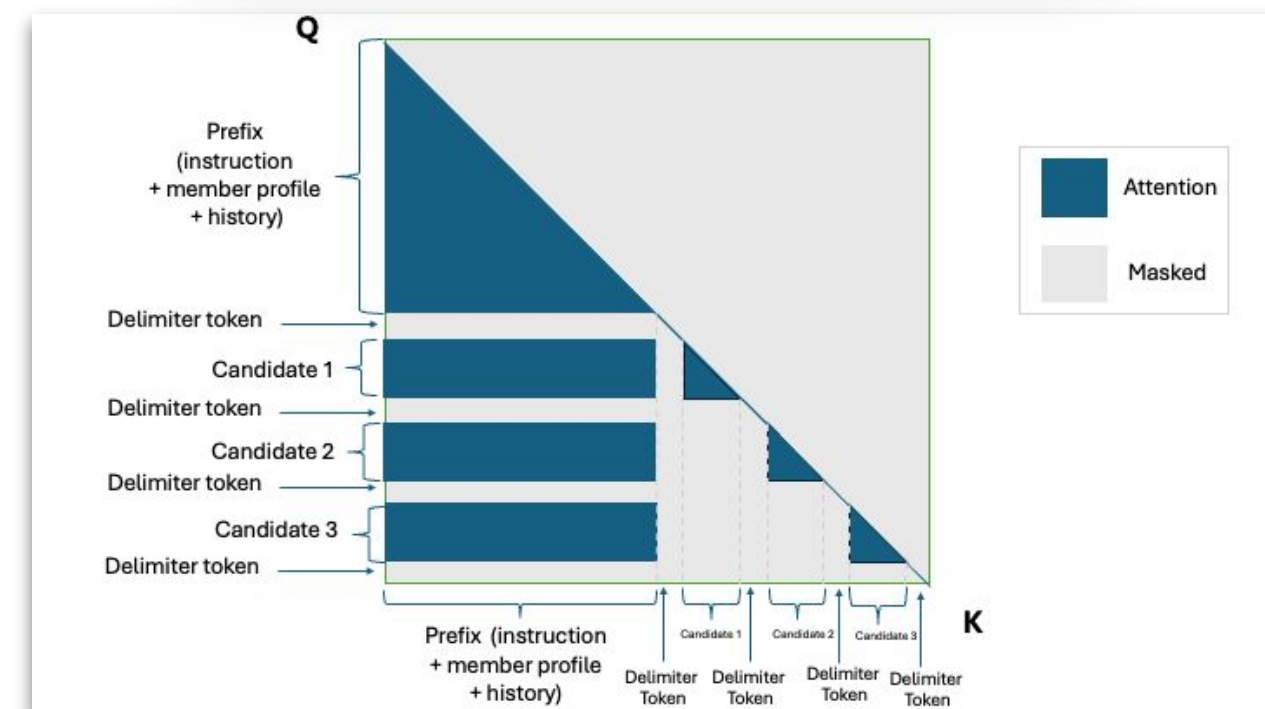
Positions: 1 2 3 4 5 6 7 8

- 2 Token 8 (Item 2) should see Query + Item 2 only (no Item 1)

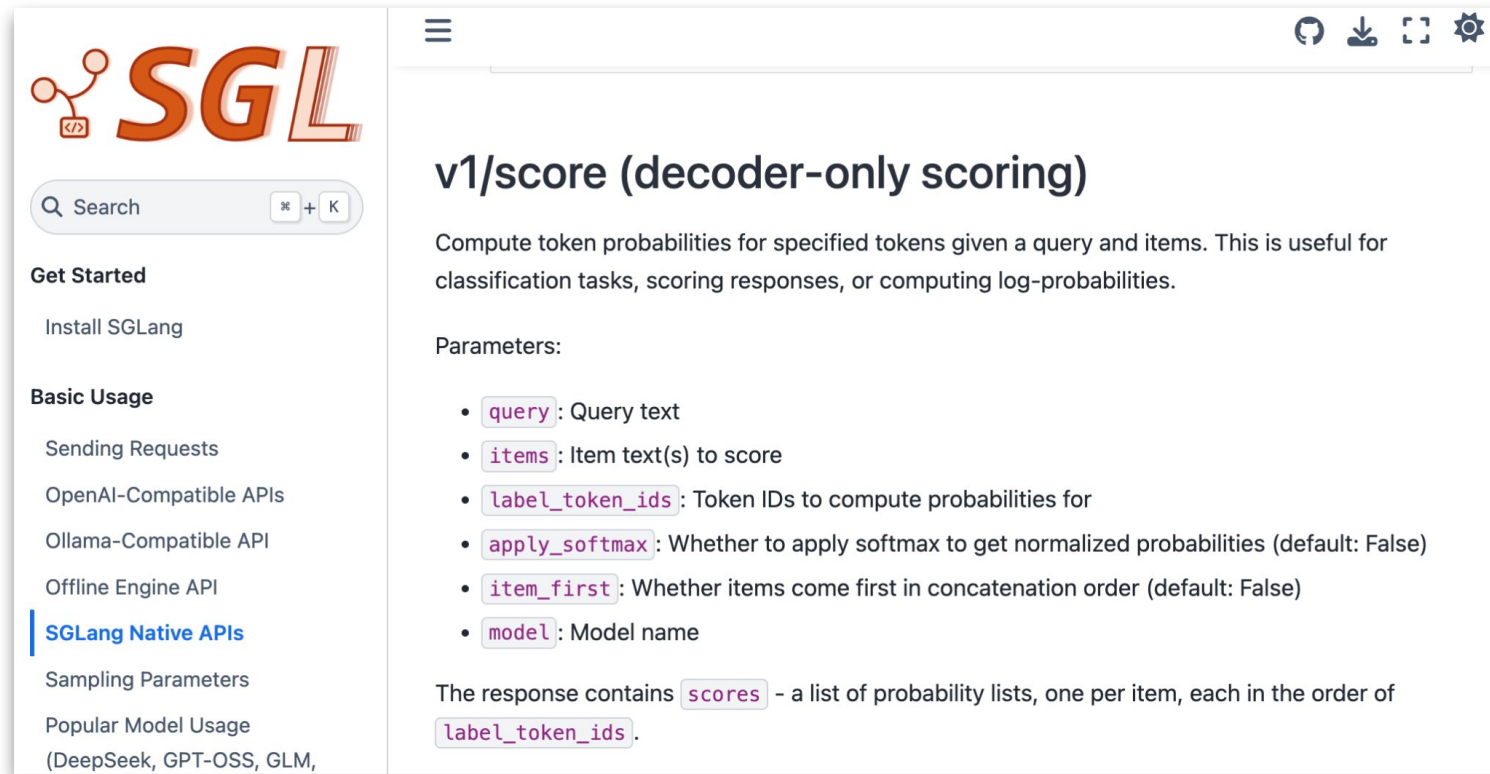
Attention mask row for token 8



Kernel item-aware attention mask ensures token 8 attends to the shared query and only its own item tokens, not Item 1.



Try It On SGLang!





The screenshot shows the SGLang documentation for the `v1/score` endpoint. The page title is `v1/score (decoder-only scoring)`. The description states: "Compute token probabilities for specified tokens given a query and items. This is useful for classification tasks, scoring responses, or computing log-probabilities." The parameters listed are:

- `query`: Query text
- `items`: Item text(s) to score
- `label_token_ids`: Token IDs to compute probabilities for
- `apply_softmax`: Whether to apply softmax to get normalized probabilities (default: False)
- `item_first`: Whether items come first in concatenation order (default: False)
- `model`: Model name

The response format is described as: "The response contains `scores` - a list of probability lists, one per item, each in the order of `label_token_ids`."

```
$ python -m sglang.launch_server \  
  --model-path /Qwen/Qwen3-0.6B/ \  
  --port 30000 \  
  --host 0.0.0.0 \  
  --chunked-prefill-size -1 \  
  --enable-torch-compile \  
  --dtype float16 \  
  --max-prefill-tokens 30000 \  
  --mem-fraction-static 0.3 \  
  --enable-dynamic-batch-tokenizer \  
  --disable-radix-cache \  
  --attention-backend flashinfer \  
  --multi-item-scoring-delimiter 151655
```

```
# Request  
$ curl -X POST "http://localhost:30000/v1/score" -H "Content-Type: application/json" -d '{  
  "query": "Is this the author of Harry Potter? Answer Yes or No for each of the following options:",  
  "items": [  
    "J.K. Rowling",  
    "Stephen King",  
    "Harry Potter"  
  ],  
  "label_token_ids": [9454, 2753],  
  "model": "/Qwen/Qwen3-0.6B/"  
}' | jq  
  
# Response  
{  
  "scores": [  
    [  
      1.025041302275678e-05,  
      5.276460429283e-06  
    ],  
    [  
      0.00043153568013765484,  
      0.0008785630681038556  
    ],  
    [  
      1.511921443022414e-08,  
      1.1927074790134633e-07  
    ]  
  ],  
  "model": "/Qwen/Qwen3-0.6B/",  
  "usage": {  
    "prompt_tokens": 31,  
    "total_tokens": 31,  
    "completion_tokens": 0,  
    "prompt_tokens_details": null,  
    "reasoning_tokens": 0  
  },  
  "object": "scoring"  
}
```

Welcoming contributions! Join us in  #prefill-only (Slack)
Roadmap:  [sgl-project/sglang/issues/15344](https://github.com/sgl-project/sglang/issues/15344)

Acknowledgement



verl: Volcano Engine Reinforcement Learning for LLMs



Thank You.



Thank You.

