Scaling Up Large Language Models Serving Systems For Semantic Job Search

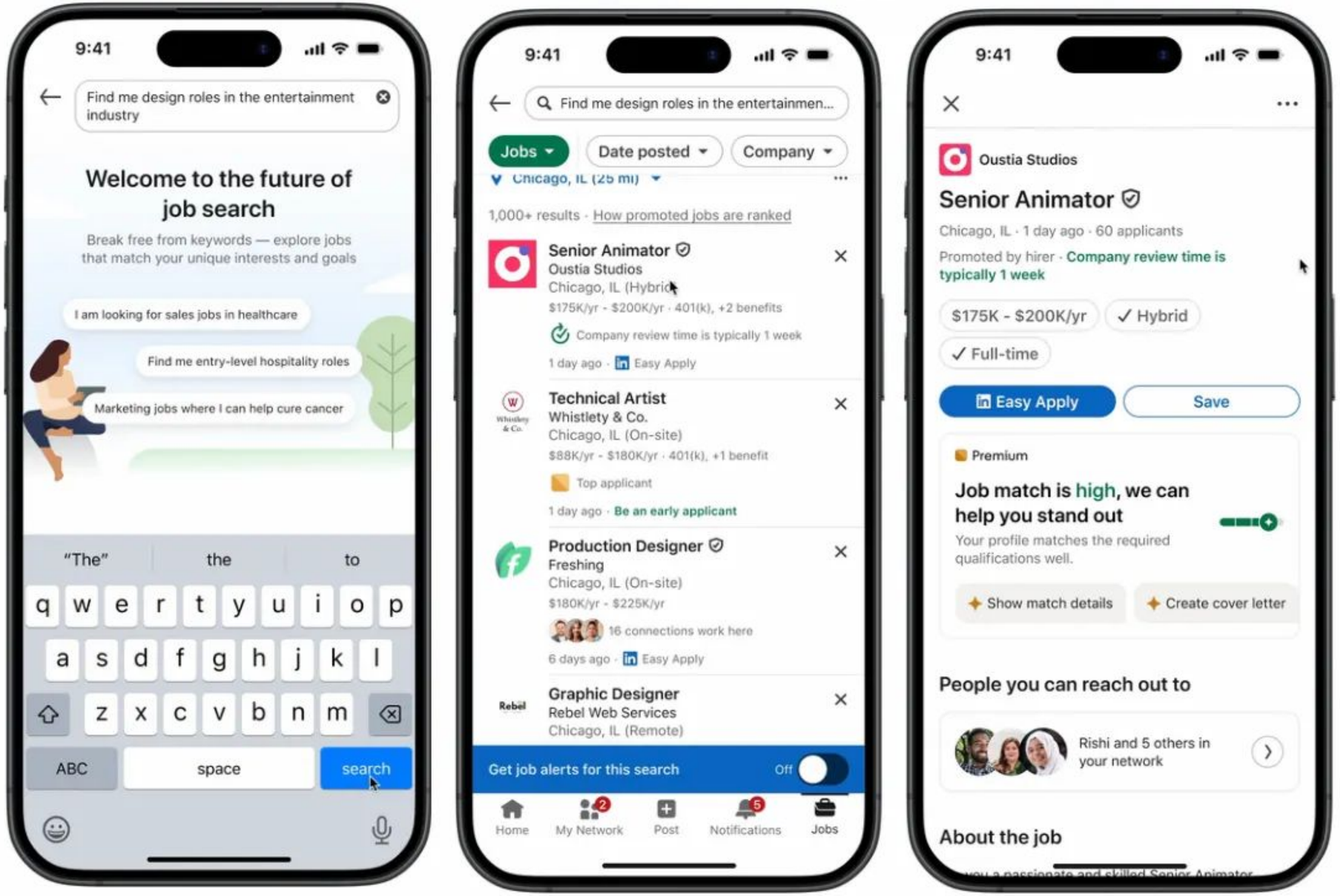
Presenters: Zhipeng Wang, Sundara Raman Ramachandran

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

May 2026

LinkedIn

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*")

+

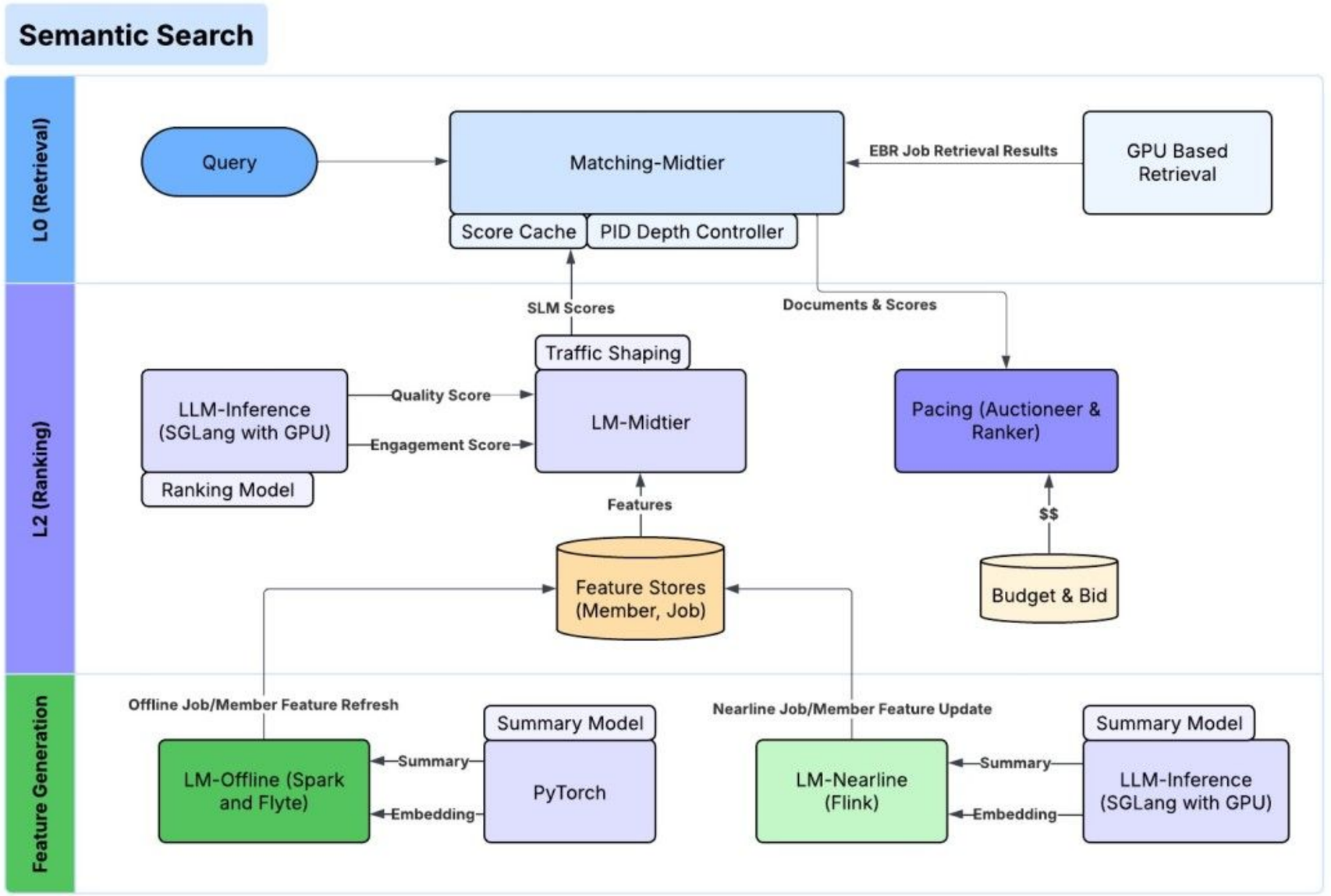
Item Description (e.g. Job Description)

↓ **LLM Ranker**

Single Label Token (Yes/No, e.g. Apply, Click etc.)

↓

Probabilities (e.g. p_{apply}, p_{click})

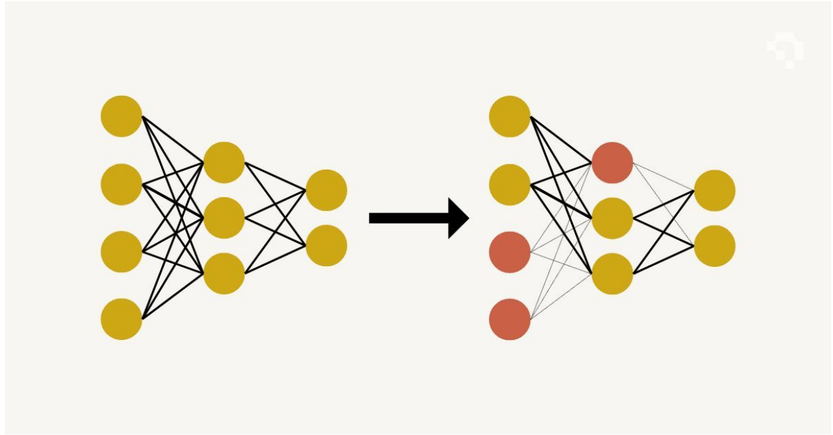


Optimizing LLM Serving Stack for Semantic Job Search

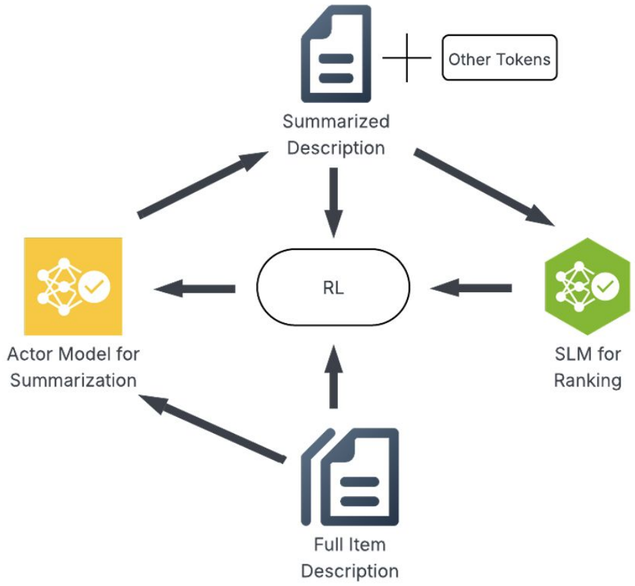
Need to serve millions of requests per second, throughput is definitely a bottleneck

*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



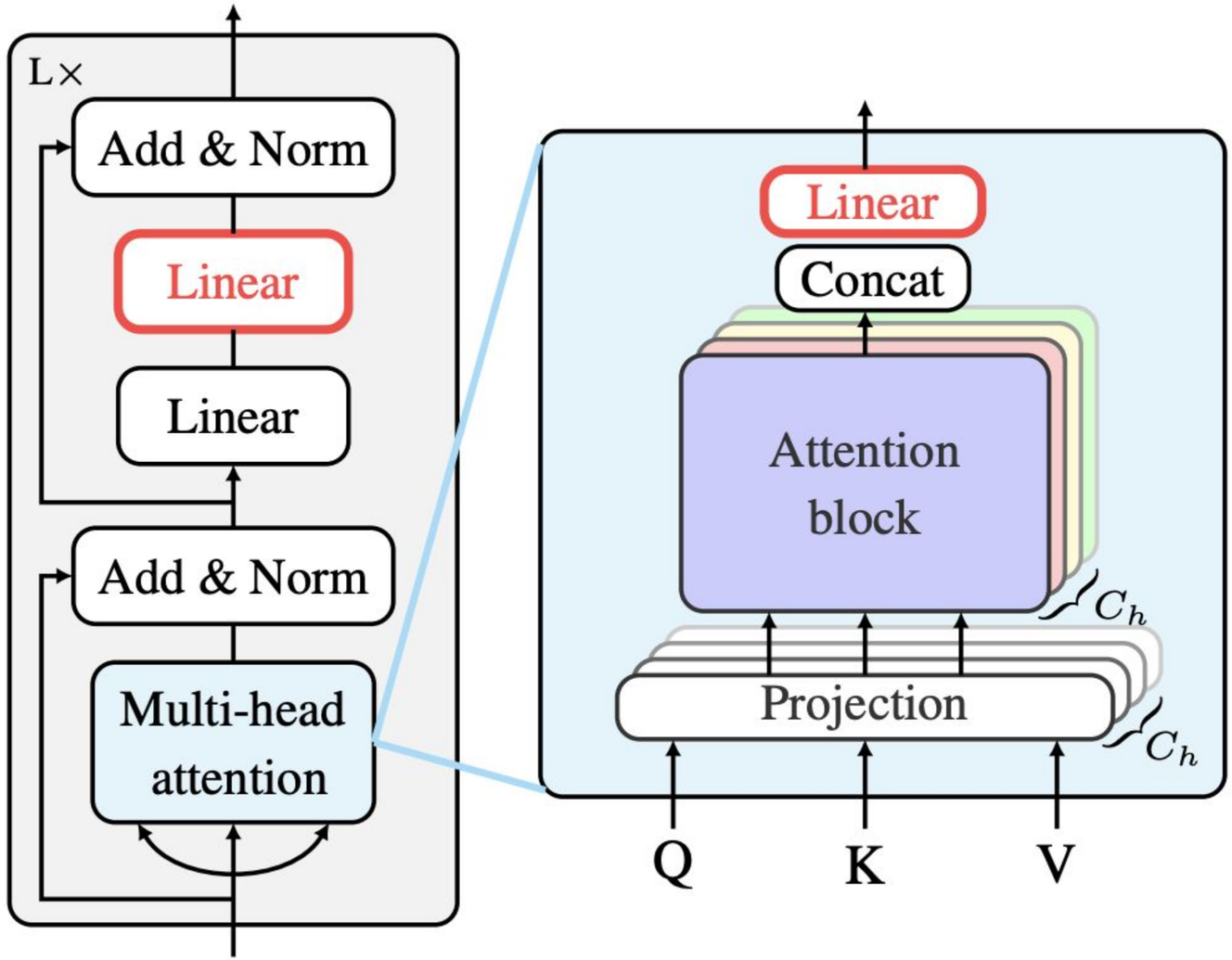
Serving Infra Optimization

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)



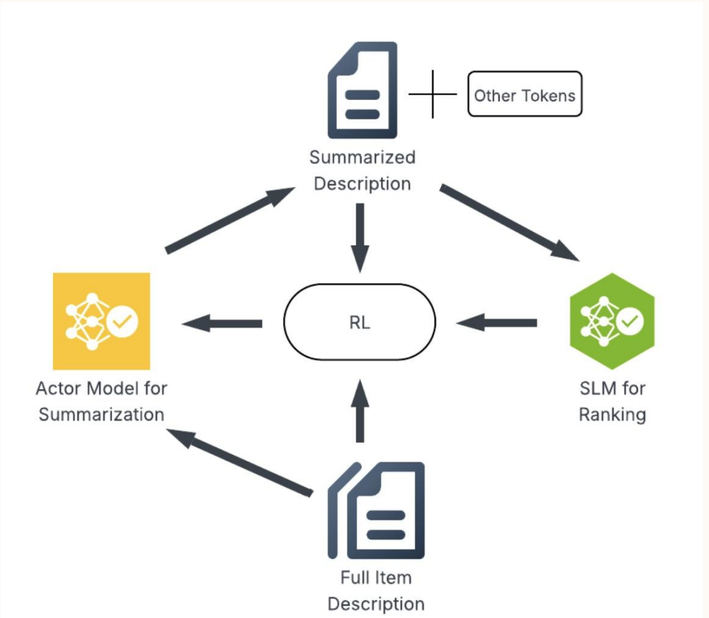
Model	NDCG@10 Change
Unpruned	-
50% MLP Pruning	−0.0095
50% MLP Pruning + SFT	−0.0046

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

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



Model Size	SFT on summarized	Evaluation on summarized	NDCG@10
600M	✗	✗	0.8950
375M	✗	✓	0.8786
375M	✓	✓	0.8788

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

Two types of length penalties

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

$$P_2(L_o, L_c) = -wr^2$$

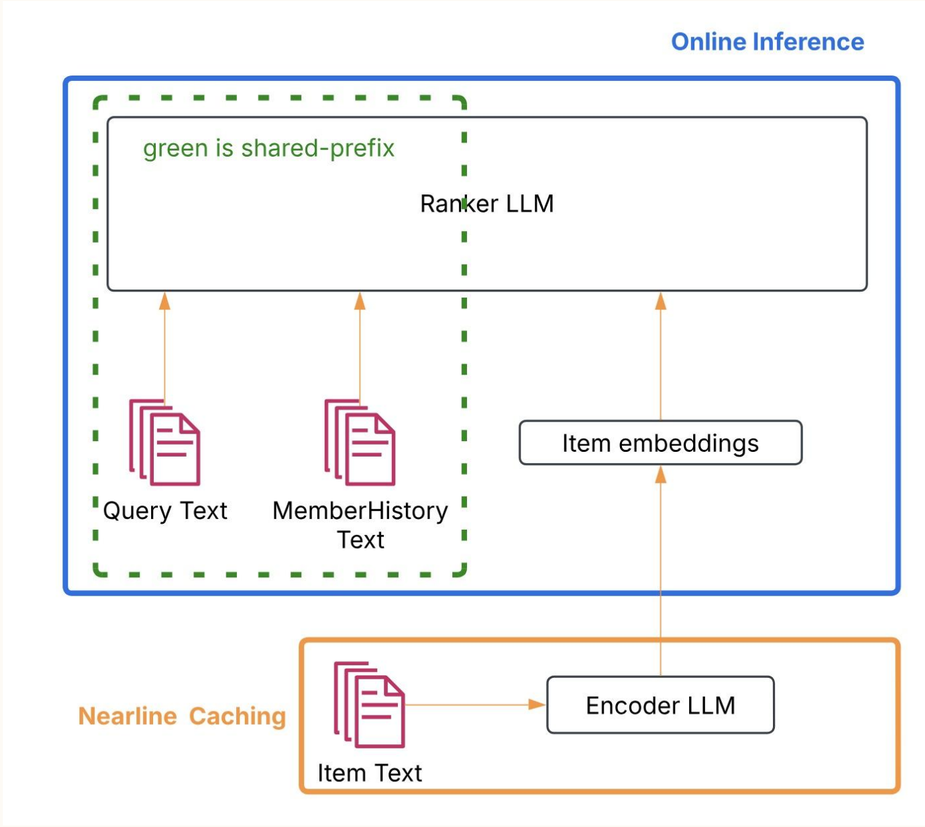
The RL training pipeline is built on top of verl and we explored GRPO, DR. GRPO and GSPO etc.

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	$\boldsymbol{P_2}$	$\boldsymbol{w = 0.4}$	**−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

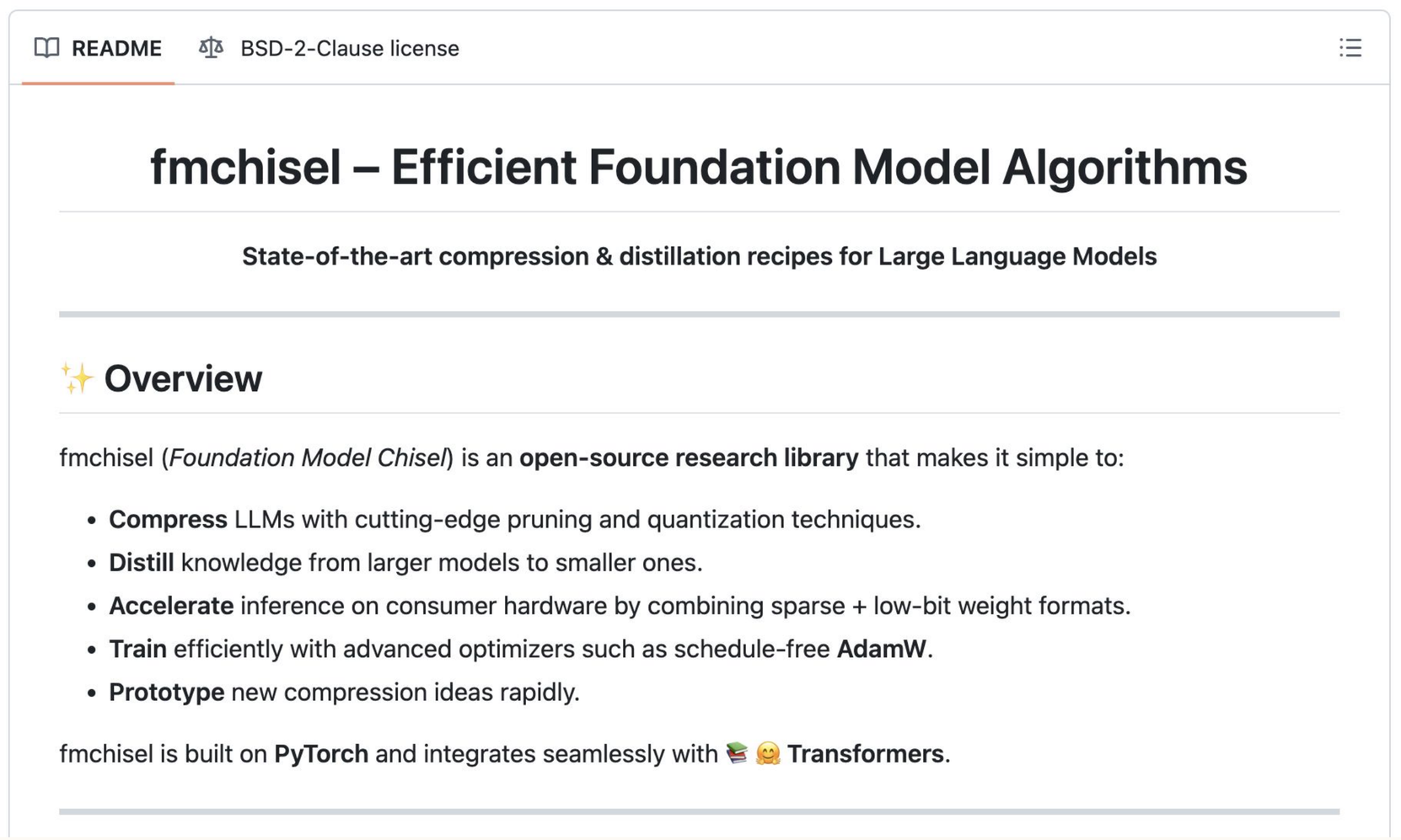


MixLM: High-Throughput and Effective LLM Ranking via Text-Embedding Mix-Interaction, arXiv:2512.07846 (KDD 2026, to appear)

- 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

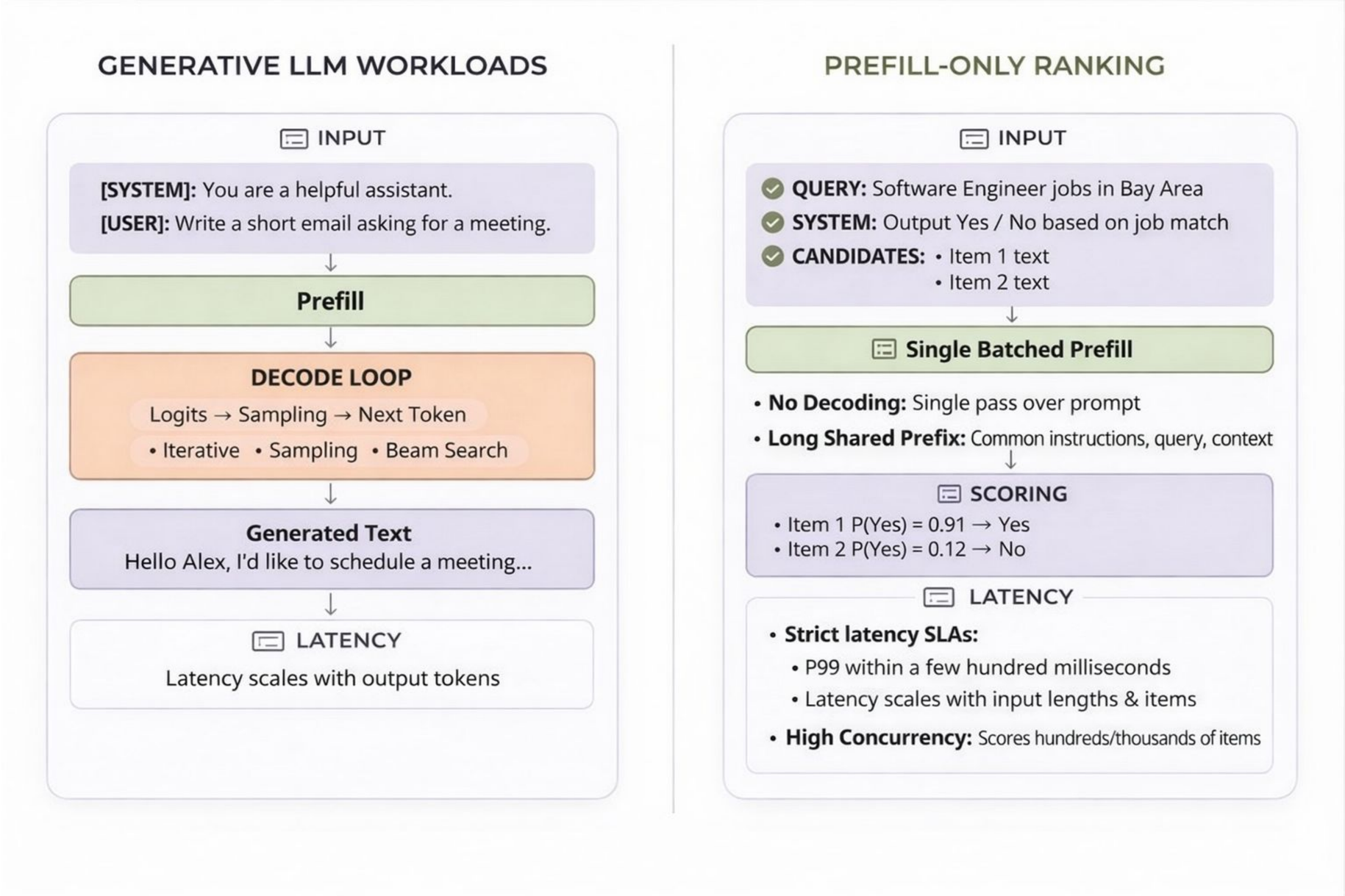
https://github.com/linkedin/fmchisel



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

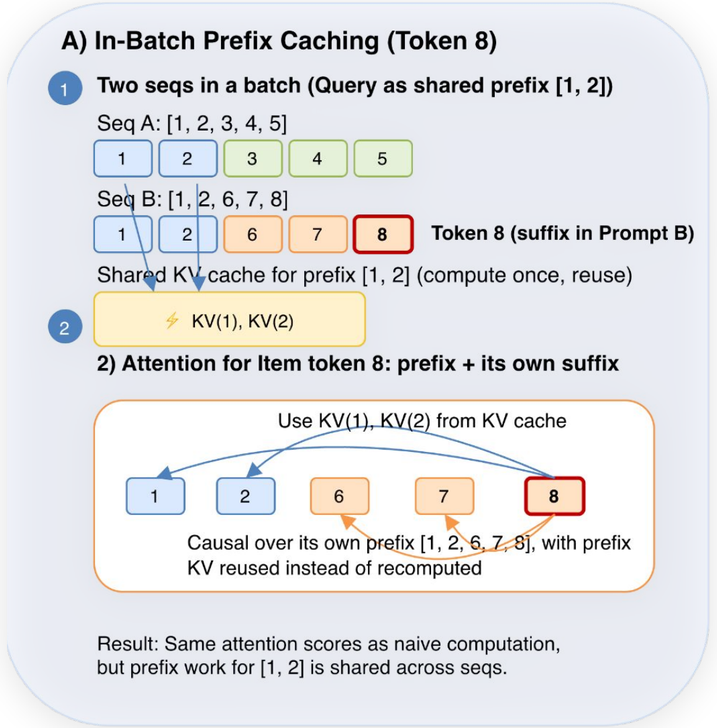
⚙️ How It Works

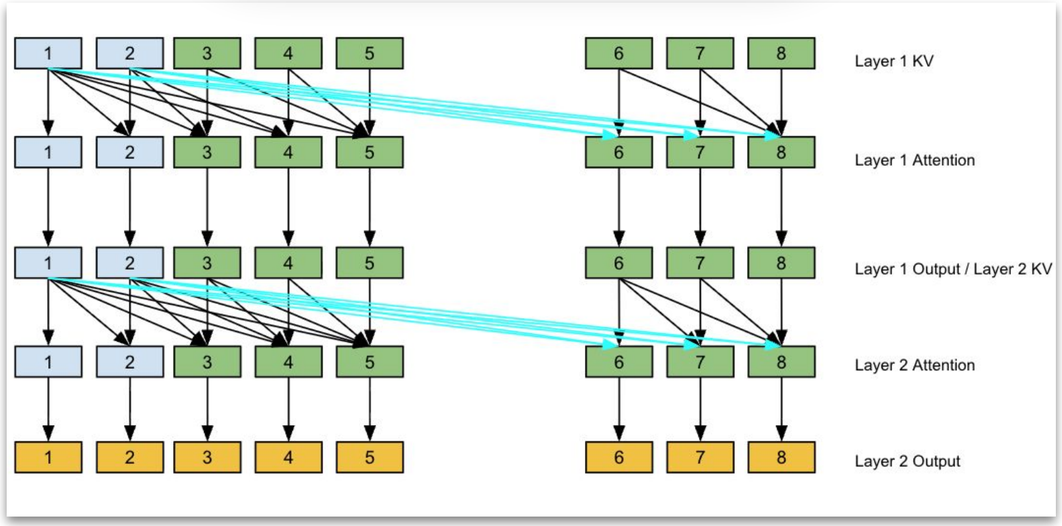
- Compute prefix KV from first prompt
- Reuse prefix KV for remaining prompts in same batch
- For suffix tokens, merge attention:
  - Prefix attention (attend to shared KV)
  - Suffix attention (causal within item)
  - Combine via log-sum-exp

✔ Numerical correctness preserved

✔ No change to model semantics

✔ Single forward pass





Reusing Query Work
Multi-Item Scoring

⚙️ How It Works

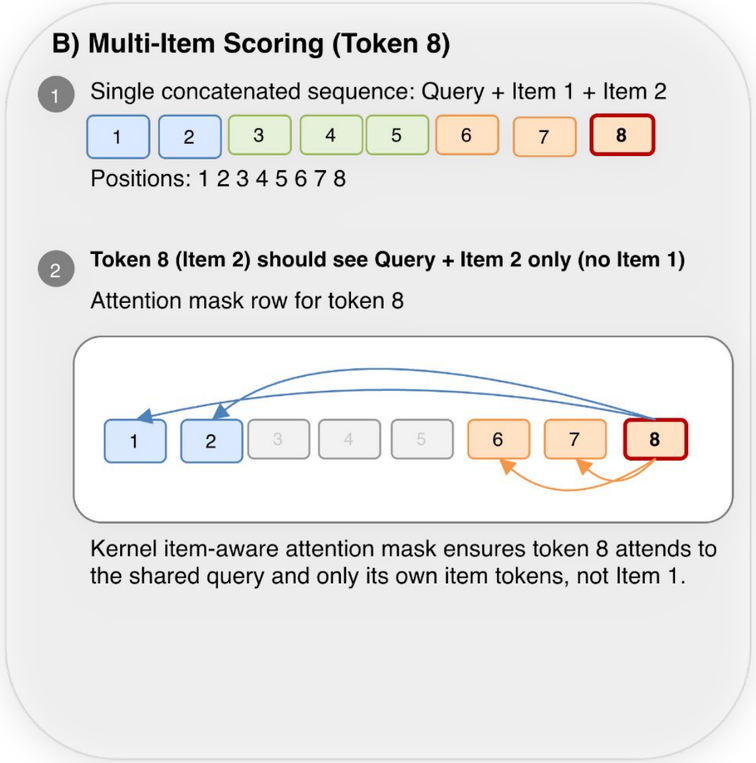
- Concatenate all items into one sequence

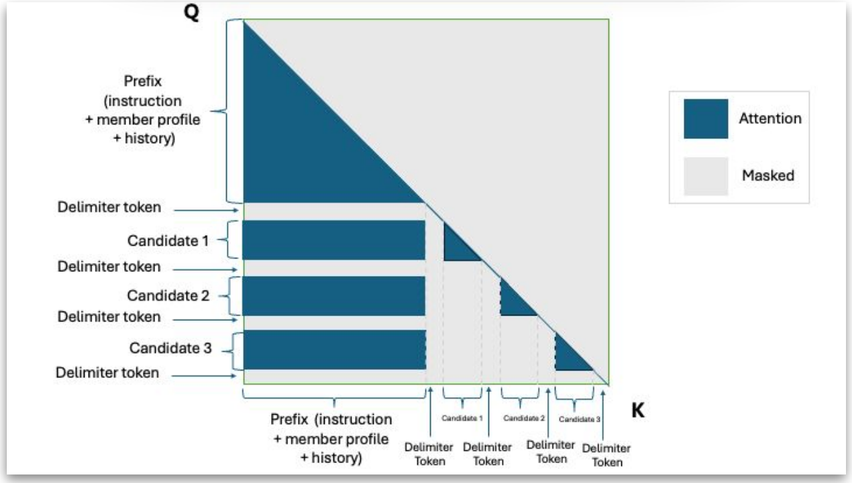
<Query><DELIM><Item 1><DELIM><Item 2>...<DELIM><Item N><DELIM>

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

Callouts

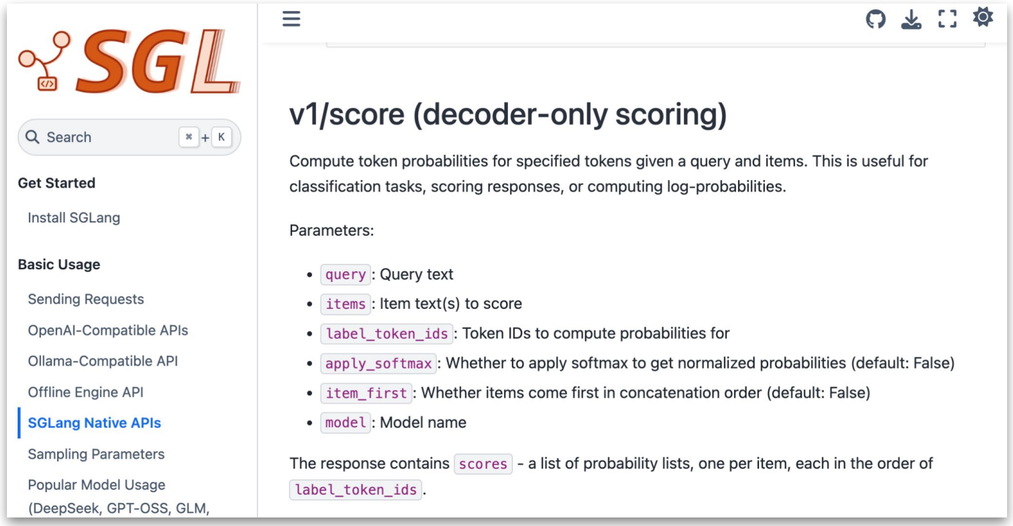
✔ Single forward pass
✔ Prefix reused naturally via concatenation
✔ No cross-item attention





sglang/pull/10979, flashinfer/pull/1015

Try It On SGLang!




```
$ 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

Acknowledgement

ByteDance | Seed

verl: Volcano Engine Reinforcement Learning for LLMs

PyTorch Foundation

Thank You.

Thank You.