SCALING UP LARGE LANGUAGE MODELS SERVING SYSTEMS FOR SEMANTIC JOB SEARCH

Kayhan Behdin [1 *] **Qingquan Song** [2 *] **Sriram Vasudevan** [1 *] **Jian Sheng** [1 *] **Xiaojing Ma** [1 *] **Z Zhou** [1 *] **Chuanrui Zhu** [1] **Guoyao Li** [2] **Chanh Nguyen** [1] **Sayan Ghosh** [1] **Hejian Sang** [1] **Ata Fatahi Baarzi** [1] **Sundara Raman Ramachandran** [1] **Xiaoqing Wang** [1] **Qing Lan** [1] **Vinay Y S** [1] **Qi Guo** [2] **Caleb Johnson** [1] **Zhipeng Wang** [1] **Fedor Borisyuk** [1]

ABSTRACT

Large Language Models (LLMs) have demonstrated impressive quality when applied to predictive tasks such as relevance ranking and semantic search. However, deployment of such LLMs remains prohibitively expensive for industry applications with strict latency and throughput requirements. In this work, we present lessons and efficiency insights from developing a purely text-based decoder-only Small Language Model (SLM) for a semantic search application at LinkedIn. Particularly, we discuss model compression techniques such as pruning that allow us to reduce the model size by up to 40% while maintaining the accuracy. Additionally, we present context compression techniques that allow us to reduce the input context length by more than 10x with minimal loss of accuracy. Finally, we present practical lessons from optimizing the serving infrastructure for deploying such a system on GPUs at scale, serving millions of requests per second. Taken together, this allows us to increase our system's throughput by 10x in a real-world deployment, while meeting our quality bar.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated impressive predictive performance in a variety of industrial tasks. Among such tasks, LLMs have been particularly successful when applied to relevance ranking for semantic search applications, where the goal is to determine how relevant an item is to a user's query. Recent work (Shi et al., 2025; Qin et al., 2024; Meng et al., 2025; Hou et al., 2024) has shown LLMs enjoy strong semantic understanding, allowing them to be powerful relevance judges. Despite their impressive performance, serving such LLMs in an industrial setting remains challenging. This is due to the fact that LLMs can introduce unacceptable latency to real-world systems, or require significant compute resources to meet the throughput goals of the system. Therefore, improving the efficiency of LLMs for ranking and relevance remains of high interest.

In this work, we study a real-world semantic search system from an efficiency perspective. At the heart of this system, we utilize a text-based decoder-only Small Language Model (SLM) that scores the relevance of items and queries. In particular, we discuss the techniques used to improve the efficiency of this ranker model to meet our strict latency and throughput requirements, while ensuring the quality remains high. We present, in details, the lessons we learned from deploying our SLM-powered semantic search system to a large-scale professional social network.

To be more specific, we explore several optimization and compression techniques, with the goal of improving the system's throughput. In particular, we explore:

1. **Model Compression via Pruning**: Pruning is a model compression technique to obtain smaller models, by removing certain redundant structures from the model. We use model pruning to obtain high-quality smaller models that enjoy higher inference throughput.

2. **Item Description Summarization**: Given that the complexity of the self-attention mechanism scales quadratically with the input context length, we design novel summarization techniques, where we create summaries of item descriptions offline using another language model trained with reinforcement learning. These summaries are then used by the SLM ranker in the online pipeline, allowing for faster online inference.

3. **Serving Infrastructure Optimization**: Our semantic search workload operates in a regime that is different from common LLM use cases, for example, in chatbots. As we discuss, we serve a prefill-only use case, with extremely high rate of Requests Per Second (RPS).

[*]Equal contribution [1]LinkedIn [2]Work done at LinkedIn. Correspondence to: Zhipeng Wang <zhipwang@linkedin.com>.

Proceedings of the 9^{th} MLSys Conference (Industry Track), Bellevue, WA, USA, 2026.

We share several lessons from optimizing our online and streaming serving stack, including lessons from optimizing the SGLang (Zheng et al., 2024) inference engine for high RPS prefill-only workloads.

1.1 Related Work

Several papers have studied compressing foundation models into smaller ones; Examples include Muralidharan et al. (2024); Sreenivas et al. (2024) which study model compression techniques such as pruning and knowledge distillation for foundation models. In another work, Gunter et al. (2024) explore several techniques such as quantization and distillation to improve the efficiency of a foundation model. We also refer to Zhu et al. (2024) for a survey of model compression techniques.

Lengthy input to language models reduces inference speed, increases memory requirements and also negatively impacts user experiences, especially in a real-world ranking system. Multiple efficient methods have been proposed for prompt compression such as Nano-Capsulator (Chuang et al., 2024) and Gisting (Mu et al., 2023). (Li et al., 2025) provides a comprehensive survey of prompt compression for large language models.

We note that, however, our work is different from the aforementioned ones. In our work we focus on an SLM used in a semantic search system, rather than a general foundation model. In particular, there has been a limited exploration of model compression techniques for relevance ranking applications. In addition, our context compression needs to work for a vast amount of versatile item descriptions, while keeping natural language format for interpretability. We share our learnings from applying model and context compression techniques, as well as GPU deployment lessons.

There are several notable difference between our work and the previous work studying LLMs for relevance in industrial applications. Wang et al. (2024) study using an LLM as a relevance judge for semantic search at Pinterest. However, instead of deploying the LLM online, they distill it to a smaller feed-forward neural network, due to the difficulty of serving LLMs at scale. In contrast, as a result of our optimizations, we directly deploy an SLM online, which as we discussed, comes with practical challenges. This further signifies the importance of the techniques we present in this paper. Similarly, Dey et al. (2025) discuss distilling an LLM relevance judge into a non-LLM architecture. In another example, Shang et al. (2025) discuss distilling an LLM into a BERT-style model for a relevance use case. This is in contrast to our work where we use a decoder-only model. Additionally, certain features such as item description are dropped from their online serving stack due to efficiency reasons, while we implement summarization techniques that allow us to include such long context features in our online system. This is another indication of the importance of the context compression methods we develop.

Finally, we emphasize that our cross-encoder ranking system operates entirely in natural language, distinguishing it from ID- or feature-based systems such as Zhai et al. (2024). As discussed above, however, scaling such a natural-language-based system remains challenging due to the growing length of prompts. This further demonstrates the significance of our work.

2 Semantic Job Search

LinkedIn's Semantic Job Search introduces an AI-driven approach to job discovery by shifting from keyword matching to intent understanding. The system leverages natural language input to capture user goals and preferences, reducing reliance on exact lexical overlap between queries and job postings. This semantic representation enables more robust retrieval, addressing vocabulary mismatches while offering a search experience that better reflects how members naturally describe career aspirations.

The retrieval layer begins with a query understanding service that parses the user's input, generates a query embedding, and runs embedding-based retrieval (EBR) on CUDA-accelerated GPUs with exhaustive vector search (Juan et al., 2025) to collect a broad pool of candidate jobs. In the ranking layer, these candidates are refined by a Cross-Encoder SLM hosted on the SGLang, which integrates search query with job and member features to produce quality scores for final ranking. We did not include any additional re-ranking layers between retrieval and SLM ranking because, as we demonstrate, SLM ranking satisfies serving quality and throughput requirements. To keep the system efficient and scalable, the ranking layer incorporates score caching, a ranking depth controller to regulate how many candidates proceed through deeper ranking, and traffic shaping to smooth request bursts—all of which optimize load management and improve search outcomes. The features and summarized job descriptions used by the SLM are precomputed in a hybrid pipeline: an offline system with Spark and Flyte for large-scale batch inference, and a nearline system with Flink for low-latency updates. These are persisted in a distributed storage system, and fetched at request time with very low latency. Finally, the auction layer applies budget and pacing logic to balance user relevance, engagement, and business objectives, ensuring both member satisfaction and marketplace health with balanced recall and precision. An overview of the overall system is shown in Figure 1.

A product policy specifies how to grade the relevance of each (query, job) pair. Concretely, we train a 7B LLM to generate 5-point graded scores along several dimensions, together with rationales, such that it aligns with the product

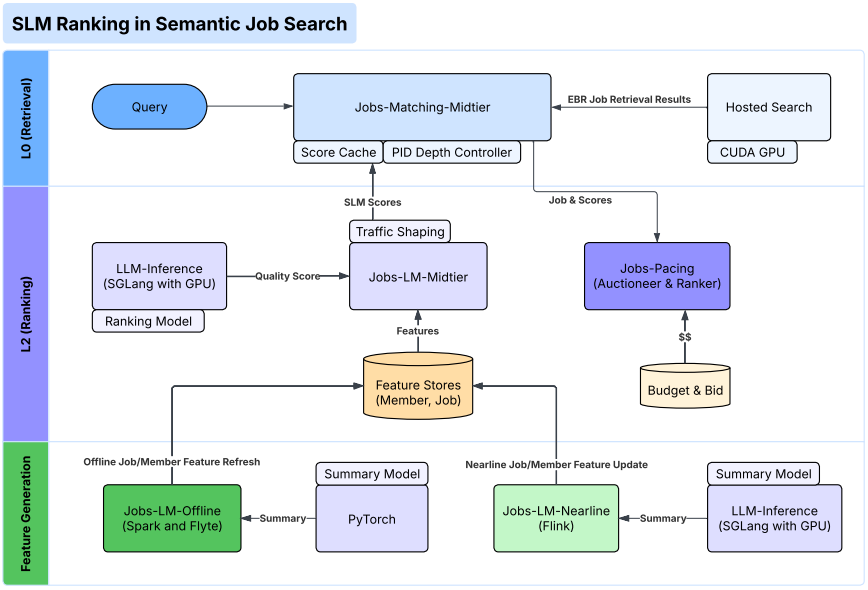

Figure 1. Overall pipeline of LinkedIn's Semantic Job Search.

policy. These dimension-wise grades are aggregated into a final label for the pair. The retrieval and ranking systems are then distilled from this teacher so that the end-to-end stack optimizes the Normalized Discounted Cumulative Gain at the tenth position (NDCG@10) using teacher-provided labels.

In this work, we focus on the ranking layer of the system. The ranking stage employs an SLM to score the relevance of a user's search query q to each retrieved job i. This SLM is a decoder-only model. We denote the structured job attributes - title, company, location, employment type, and remote-work eligibility - by metadata_i, and the free-text job description by desc_i. For each $(q,\ i)$ pair, we construct a prompt

$$\text{prompt}(q, i) = \text{system prefix}, q, \text{metadata}_i, \text{desc}_i, \text{suffix}, \tag{1}$$

where system prefix/suffix include chat-template tags and instructions to decide whether the item matches the query. Passing this prompt through the decoder yields logits for the next token. Let $\text{logit}_{\text{yes}}, \text{logit}_{\text{no}}$ denote the logits for the tokens "yes" and "no" corresponding to the last input token, respectively. Following prior work (Wang et al., 2024; Zhang et al., 2025), we compute

$$(p_{\text{yes}}, p_{\text{no}}) = \text{Softmax}(\text{logit}_{\text{yes}}, \text{logit}_{\text{no}}) \tag{2}$$

which yields probabilities used to rank items by relevance.

SLM Training uses a multi-stage pipeline. First, we distill the 7B teacher into a 0.6B model to obtain a smaller model that generates graded labels with rationales. Next, we map the teacher's ordinal grades to $(p^*_{\text{yes}}, p^*_{\text{no}})$ "soft labels," and perform Supervised Fine-Tuning (SFT) to minimize the Kullback–Leibler (KL) divergence between the teacher's soft labels and the SLM's predictions, turning the SLM from a reasoning model into a binary classifier:

$$\text{loss} = KL((p^*_{\text{yes}}, p^*_{\text{no}}) \| (p_{\text{yes}}, p_{\text{no}})) \tag{3}$$

with $\text{KL}(p\|q) = \sum_i p_i \log(p_i/q_i)$.

We train on 200k examples for up to 5 epochs using the prompt in (1). The prompt is dominated by desc_i, whose length varies widely (median ~ 900 tokens; max > 2100). We truncate desc_i so that the total prompt length does not exceed 2048 tokens during both training and inference.

Evaluation uses a holdout set labeled by the teacher. The SLM ranks candidates by p_{yes}, and we report NDCG@10.

To meet our job search traffic across multiple product facets, our ranking system has to be able to score 3.15M items per second, making the efficiency of the ranker of utmost importance.

3 Discussion of Compression Techniques

3.1 Model compression via structured pruning

To improve the throughput, we explore model compression via pruning. Model pruning (LeCun et al., 1989; Hassibi et al., 1993; Benbaki et al., 2023) is a model compression technique where some model weights or components are removed to obtain smaller models, with a smaller computational footprint. Model pruning has been successfully applied to LLMs (Frantar & Alistarh, 2023; Sun et al.; Meng et al., 2024), allowing model compression with a small loss

of quality. We note that, however, most existing papers do not study model pruning in the context of recommender systems. Behdin et al. (2025) study model pruning for sequential recommender systems. Our work is different where we study a deployed semantic search system and focus on a cross-encoder ranking system.

In this work, as we are interested in deploying our model on GPUs, we focus on structured pruning. In structured pruning, certain model components (neurons, layers, attention heads) are removed from the model, resulting in models that are smaller with fewer parameters. Such pruned models can then be deployed without requiring specialized hardware or kernels, leading to improved throughput and latency. This is in contrast to unstructured pruning, where inference acceleration might only be possible on certain hardware. Specifically, in our work, we consider pruning of:

- Hidden neurons in Feed-Forward MLP layers
- Whole transformer blocks.

We use OSSCAR (Meng et al.) for pruning hidden neurons in MLP blocks. We also experiment with removing complete transformer blocks in the model—we present our results in Section 4.1. These pruning approaches result in smaller models that can be accelerated on any hardware. In particular, removing hidden neurons in MLP layers reduces model's intermediate size, while removing transformer blocks results in a model with a smaller number of hidden layers. After performing pruning, we fine-tune the pruned model to recover any accuracy lost due to compression.

3.2 Context compression via summarization

Large Language Models (LLMs) have demonstrated impressive capabilities in contextual learning, leverage more complex and sophisticated input to achieve superior performances. However, longer prompt length leads to higher latency and larger memory consumptions. In particular, in our use case of SLM, the time-to-first-token (TTFT) latency increases quadratically with regard to input length, limiting its applicability to serve real-time traffic for billions of users.

We explore context compression techniques to improve further latency and throughput, while maintaining model quality as much as possible. Prompt / context compression has gained significant research interest (Li et al., 2025) in recent years. For example, Gisting (Mu et al., 2023) trains an LM to compress prompts into smaller sets of "gist" tokens which can be cached and reused during inference. However, this is usually used for general system prompt / task descriptions, and cannot be generalized to a vast amount of heterogeneous item descriptions. Nano-Capsulator (Chuang et al., 2024) compresses original prompts into natural language

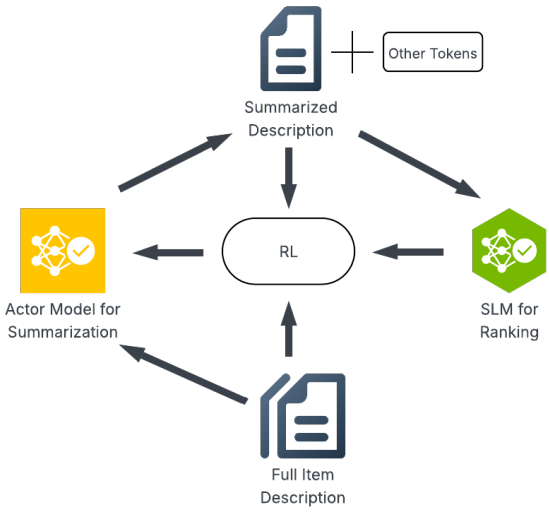

Figure 2. **Overall pipeline of context summarization with RL.** We start with full item description, which is processed by the actor model to generate a summarized version. The summarized description is then combined with other tokens such as metadata before feeding into the SLM. The reward to the RL is calculated using the output of the SLM, coupled with the length of full and summarized description to update the parameters of the actor model. The parameters of the SLM are kept frozen throughout the process.

formatted prompt while maintaining the prompt utility and transferability. It uses a semantics preserving loss coupled with a reward function featuring length constraints. In our use case, text descriptions can be verbose and contain information that isn't useful for ranking, which cannot be distinguished with semantic loss alone. It is worth noting that the papers discussed here do not specialize to text summarization for relevance ranking, where we study a highly fine-tuned model rather than a base model.

We propose to fine tune a language model for generating summarizations based on raw context with Reinforcement Learning (RL). Given our two objectives on 1) shorten context length and 2) maintain model quality, our reward signal is constructed with two components respectively: 1) an output sequence length penalty and 2) KL divergence loss between the SLM outputs $(p_{\text{yes}}, p_{\text{no}})$ on summarized context and raw context. Equation (4) shows the reward signal, where the subscript denotes summarized and raw context respectively, and w is a hyper-parameter controlling the relative importance from length penalty.

$$\text{reward} = -KL(p_{\text{sum}} \| p_{\text{raw}}) - w(\frac{len_{\text{sum}}}{len_{\text{raw}}})^2. \quad (4)$$

We are not aware of any previous work on RL-based summarization in the context of relevance ranking. However, we note that the approach we proposed here shares similarities

Table 1. The results for pruning hidden neurons in MLP layer via OSSCAR, with and without SFT after pruning. The pruned model has around 460M parameters, compared to the the unpruned version with 600M.

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

with reinforcement learning from verifiable rewards (Guo et al., 2025; Lambert et al., 2024), and recent work on reinforcement learning from performance / user feedback (Jiang et al., 2025; Han et al., 2025). The diagram in Figure 2 is an overview of the whole pipeline.

Our RL training pipeline is built on top of verl (Sheng et al., 2025), and we use GSPO (Zheng et al., 2025) as the optimization algorithm. We present experiment results in Section 4.2, where we show that our compression method can achieve over 10X context compression ratio with only 2% quality drop. We also include comprehensive ablation studies regarding the necessity of reinforcement learning and length penalty design.

4 EXPERIMENTAL RESULTS

4.1 Pruning Experiments

In this section, we present our model pruning experiments and the insights from our experiment. The experiments in this section use prompts with full item description (that is, no summarization is not applied in this section). In the next sections, we discuss integrating summarization and pruning.

We start by studying the effect of pruning hidden neurons in the MLP layers of the model. To this end, we start from a 0.6B model that has been fine-tuned on our relevance data. Next, we remove 50% of hidden neurons in all MLP layers via OSSCAR. Pruning methods such as OSSCAR often use a small dataset (known as the calibration data) to choose which neurons, for example, can be removed from the model. Following the prior work (Behdin et al., 2025), we use in-domain data as for calibration data, using around 40M tokens from our training prompts. Additionally, after pruning, we perform SFT to further recover the accuracy lost after pruning. The results for pruning 50% of hidden neurons in MLP layers is presented in Table 1.

We observe that even when pruning half of the hidden neurons in the MLP layers, we observe a modest drop in accuracy (less than 1% in terms of NDCG@10). We also observe that most of this accuracy drop is recovered after performing SFT on the pruned model. As the quality drop after pruning is small, this SFT step is rather light. Specifically, we require around 16 GPU (H-100) hours for this stage, which is negligible compared to the cost of model pretraining. This signifies the desirability of model pruning. After pruning, we obtain a model with around 460M, a 25% reduction in model size.

Table 2. Results for pruning transformer blocks from the model. We remove a transformer block at a time, and evaluate the model's quality. We see that removing last layers has a smaller effect on model's quality.

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

Table 3. Integrating MLP and transformer block pruning. We consider 50% pruning of MLP hidden neurons, and removing last few transformer blocks of the model. We include SFT steps after each pruning step.

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

Next, we explore how to remove whole transformer blocks from the model. To this end, we remove one transformer block at a time, and then evaluate the resulting model's quality. We show the results in Table 2. We observe that the model quality seems to be least sensitive to the removal of last few layers, while the first and middle layers seem to be most important and need to be preserved.

Combining our experiments from Tables 1 and 2, we combine MLP pruning and transformer layer pruning into one pipeline, to obtain even smaller models. To this end, we first follow the same recipe from Table 1 (with SFT). Then, we remove the last few transformer layers from the model to obtain a smaller one. Next, we run SFT on top of this model that has undergone two rounds of pruning. The results for this case are presented in Table 3. Based on our results, we are able to reduce the model size by 45% while losing less than 1% of the quality, allowing us to significantly increase our inference throughput with a marginal quality loss.

Table 4. Context compression with prompt engineering.

APPROACH	NDCG@10	COMPRESSION
NO ITEM DESCRIPTION	-9%	-100%
STOP WORD REMOVAL	-0.5%	-16%
"SUMMARIZE" PROMPT	-2.5%	-52%
YAML PROMPT	-2.7%	-74%
"KEY PHRASES" PROMPT	-1.5%	-61%

4.2 Context Compression Experiments

As mentioned in Section 2, complete item descriptions have a median length of 900 tokens and a maximum of over 2300. This results in over 94% of the SLM prompt being comprised of just the description, and around 10% of all prompts being truncated due to the 2048 token budget. Excluding the description entirely results in a major performance drop (Table 4), indicating that there is significant value in retaining this field for the relevance prediction task.

We therefore utilize a more powerful 1.7B LLM to summarize these descriptions, since item-specific inference can be precomputed offline and updated using streaming solutions.

4.2.1 Prompt Engineering

Our preliminary experiments combined heuristics with a pure prompt-engineering approach, using the open-weights version of this model:

- **Stop word removal:** We used heuristics to strip tokens that matched those in a special dictionary since we observed that the 1.7B model was unable to follow a "remove stop words" instruction consistently.
- **"Summarize this job description" prompt:** We prompted the model to directly output summaries, but observed shorter responses that didn't capture the job requirements very well.
- **YAML-focused prompt:** We used the model to extract facts from the job description and present it as a structured output. This worked well from an instruction-following perspective, but we observed similar results as in the previous prompt scenario.
- **"Rewrite using key phrases that job seekers might search for" prompt:** By avoiding the "summarize" keyword, the model generates longer descriptions, resulting in a smaller performance drop.

Our results are captured in Table 4. We see that a pure prompt-oriented approach is insufficient to achieve significant length compression without considerable model performance degradation.

Some additional avenues to explore in the prompt-engineering space include using DSPy (Khattab et al., 2024) to optimize the summary prompt based on rewards and constraints.

4.2.2 Reinforcement Learning

A fundamental issue with the prompt-engineering approach is that the generated summary is not aligned with the job search SLM, leading to a performance degradation. Furthermore, setting limits on the summary lengths is also error-prone with just a prompt-based approach. We therefore leverage Reinforcement Learning based approaches to tune the weights of the summary model (the actor) instead, aligning it with the SLM (reward model) while also incorporating a length penalty. RL results in a more generalized summary model compared to a Supervised Fine-Tuning approach. Furthermore, we don't need to provide example summaries in our training dataset. We reuse the simple "Summarize" prompt as the input to the actor.

For the RL algorithm, we experimented with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), GRPO Done Right (Dr. GRPO, an improvement over GRPO) (Liu et al.), as well as the more sample-efficient Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025).

We also experimented with two different length penalty formulations described in Equations (5) and (6),

$$P_1(L_o, L_c) = \begin{cases} 0, & \text{if } L_o < m \\ & \text{or } r \le \tau, \\ -w\left(\dfrac{r-\tau}{1-\tau}\right)^2, & \text{otherwise,} \end{cases} \tag{5}$$

$$P_2(L_o, L_c) = -wr^2 \tag{6}$$

where L_o and L_c are the original and compressed token lengths respectively, r is the compression ratio L_c/L_o, m is the maximum acceptable length and τ is the largest acceptable compression ratio.

The motivation for P_1 is to encourage compression only for descriptions longer than m, and to keep the compression ratio $r = L_c/L_o$ at or below a target τ for most cases. Between τL_o and L_o, the penalty increases smoothly, using a quadratic term so that longer summaries incur disproportionately larger costs. By contrast, P_2 penalizes descriptions of all lengths, biasing the actor toward shorter outputs at potential expense to SLM performance. We adopt a quadratic norm rather than a linear one because it minimizes the expected overshoot $\mathbb{E}[(r-\tau)_+]$, whereas a linear form primarily reduces the tail probability $Pr(r > \tau)$. From a throughput standpoint, the magnitude of violations above τ matter more than their mere frequency, making the quadratic choice preferable.

In our experiments, we fixed $m = 256$, $\tau = 1/3$ based on

system requirements and only tune w for various algorithm-penalty combinations. Our results are captured in Table 5. An interesting observation with P_1 is that as we increase w, we see more than 66% compression despite $\tau = 1/3$ because the asymmetric reward creates a *margin-seeking* or risk-averse effect. When w is large, any sample that exceeds τ is considerably worse than those that are below it, so the policy learns to hedge by moving well below τ. With P_2, we observe that it lags in performance compared to P_1 for less aggressive compression (a 2% NDCG drop for 89% compression, compared to $< 2\%$ for a similar compression using P_1). However, as we attempt to compress more aggressively (eg: 94%), P_2 performs better, likely because we don't zero out the penalty as the summaries get shorter, so the gradient updates are not sparse and the model still learns. Finally, coming to GSPO, we observe that it is able to deliver a better performance-compression tradeoff with P_2, as compared to Dr. GRPO.

With a maximum acceptable drop of 2% NDCG@10, we therefore chose GSPO with the P_2 length penalty and $w = 0.4$, giving us a 93% reduction in p50 and p99 item description lengths. Comparing this to the "no description" case from Table 4, we see that we can compress context worth 7% NDCG@10 into descriptions that are just 7% of their original lengths. The original item descriptions make up almost 94% of the SLM's prompt. This drops to 54% with the compressed item descriptions.

Beyond the $93\% - 95\%$ compression range, we almost always notice a precipitous drop in model performance. At these high compression rates, we believe this is due to the summary model being forced to drop information, since it can no longer capture this using the language's vocabulary. Updating the algorithm to incentivize outputs in a more token-efficient language might alleviate this challenge. Another observation is that the summary model learns to generate well-formed text, perhaps due to the reward model having been trained on complete job descriptions. We see a terse note-taking style emerge only when the penalty weight is high. Finally, since the reward model was trained on English job descriptions and the summary prompt is also in English, we observe a tendency for the trained actor to output summaries in English even when the original job description is in another language. The summaries are still valid though, due to the LLM's inherent translation capabilities.

4.3 Integrating Model and Context Compression

We combine model and context compression experiments discussed above in our final pipeline. From our experimental results, we MLP pruning via OSSCAR, and removing the last few layers of the model (Table 3). In our final pipeline, we prune 50% of the MLP hidden neurons, as well as the last 8 transformer blocks, leading to a 375M model. After pruning, we perform SFT, either using the full item descriptions or summarized item descriptions. We then evaluate the model on the evaluation data that only includes the summarized item descriptions (which is the production setup). The overall quality results are presented in Table 6. We see that after pruning and summarization, the model quality drops by less than 2%, while as we discuss, taken together, these compression techniques allow us to significantly improve the ranking throughput. Interestingly, we observe that performing SFT after pruning using the summarized data results in a marginally higher NDCG value.

5 Inference Benchmarking and Optimization

In this section, we present our results regarding the inference and deployment of our SLM in the semantic search application.

5.1 Offline Benchmarking

Before moving to an online serving setup, we perform extensive offline benchmarking to understand how model and context compression allow us to increase the system throughput. In particular, we start by benchmarking the raw inference engine, to demonstrate the benefits of our compression techniques.

To this end, we use SGLang (version 0.4.6) inference engine with NVIDIA H100 GPUs for inference (CUDA version 12.6). To simulate the online traffic, for each query we create 10 different prompts using 10 different candidate items. We then send the batch of prompts to the engine at random times for scoring, according to a Poisson process. We call each batch of 10 prompts a request.

We increase the average Requests Per Second (RPS), until the 95-th percentile end-to-end latency reaches 500 milliseconds, which is our acceptable latency. This allows us to obtain the maximum traffic throughput the engine can handle.

We consider several setups to benchmark. First, we use the (unpruned) 0.6B model, using full item description without summarization. We use our training data to form the input to the engine. This leads to a maximum prompt length of 2k tokens, with an average of 900 tokens per prompt. Next, we consider the 375M pruned model from Table 6, using the full item description with no summarization. Finally, we study the setup where we use the pruned model, as well as the summarized item descriptions for inference. Under this setup, the average prompt length reduces to 220 tokens, a 4x reduction in average prompt length. We report the throughput we can achieve per GPU in Table 7 for the cases discussed here.

Table 5. RL Experiments. We report the p99 compression here, although we observed fairly similar compression at p50 too.

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	$\boldsymbol{P_2}$	$\boldsymbol{w = 0.4}$	$\mathbf{-2\%}$	$\mathbf{-93\%}$
GSPO	P_2	$w = 1.0$	-2.2%	-95%

Table 6. Integrating model and context compression. The first row does not include any compression. "SFT on summarized" indicates that if the fine-tuning after pruning data uses the summarized item descriptions or not. Similarly, "Evaluation on summarized" indicates that if the evaluation data uses the summarized item descriptions or not. In production, we use the pruned model and summarized item descriptions for SFT and evaluation (the last row).

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

Table 7. Throughput (number of items scored per second per H100 GPU) in an offline setting. The original refers to using the 600M model with full length item description. Pruned refer to using the 375M pruned model with full length item descriptions. Pruned and Summarized refers to using the 375M model with summarized item descriptions.

SETUP	THROUGHPUT (ITEMS/SEC/GPU)
ORIGINAL	330
PRUNED	420
PRUNED AND SUMMARIZED	1530

We observe that by using a smaller model (375M parameters instead of 600M), we achieve a 27% throughput lift (from the engine perspective). Combining pruning and context compression, we see that we achieve a 4.6x throughput lift, while losing less than 2% of the model's quality (see Table 6).

Quantization Beyond model pruning and context summarization, we have also explored model quantization to improve the efficiency of our system.

More specifically, we evaluated the impact of FP8 quantization on the 0.6B model using static weight and activation quantization (W8A8). To perform quantization, we used SmoothQuant (Xiao et al., 2023). In terms of model quality, FP8 quantization generally preserves the model performance with minimal degradation. Our inference benchmarking however, reveal that we can realize an up to 10% lift in throughput. As a result, we decided against using FP8 quantization in our online serving, as the increase in throughput did not justify the additional complexity resulting from quantization. As to why we do not observe a higher throughput increase from quantization, we observe that quantization mostly reduces the memory and computational footprint of linear layers (e.g., MLP layers). As we are serving a small model, such layers might not be the inference bottleneck.

5.2 Serving Engine Optimization

Our semantic search workload differs from standard (e.g., chatbot) workloads in several aspects; First, our workload only includes a prefill phase as we only need the logits corresponding to the last input token to generate our relevance scores based on (2). Second, we work with extremely high number of requests per second, where, for example, even tokenization can become a major bottleneck. Finally, our prompts follow a specific structure, where all prompts corresponding to a query have a shared prefix that include the system instructions and the query itself. This creates opportunities to improve the system throughput through clever Key-Value (KV) cache management in the attention mechanism. Hence, we optimize our online serving pipeline to

Table 8. The effect of batch tokenization in SGLang, compared to sequential tokenization. A higher batch size improves tokenization efficiency.

BATCH SIZE	TOKENIZATION SPEEDUP
1	1.0x
2	1.7x
4	2.8x
8	5.2×

improve the throughput in a production environment. Below, we discuss some of the techniques we applied.

Batch tokenization Tokenization—the conversion from raw text to model tokens—is an often-overlooked part of inference latency. For large-scale production systems like LinkedIn's SLM ranker, tokenization can become a dominant cost at high RPS or during prefill-heavy batched requests. To address this bottleneck, we have implemented batch tokenization in the SGLang engine. In particular, we tokenize all text prompts inside a single request in one pass, instead of making several calls to the tokenizer. The tokenizer implementation typically performs vectorized work when encoding many strings at once; invoking it once per request amortizes overhead and reduces per-text tokenization time. The relative tokenization speedup (compared to a sequential setup) is reported in Table 8.

Removing the decode phase Scoring/prefill-only workloads do not require per-token sampling or iterative decoding; they only require the final token's distribution (see (2)). Running the full decode and sampling loop imposes unnecessary CPU work and memory traffic. To address this issue, we added a scoring-optimized path that skips sampling/decoding phases altogether and computes only the final token probabilities required for scoring. This happens deterministically (no sampling side effects) and avoids the per-token sampling overhead. In our experiments, we observe that for a single request (10 prompts) with a 300 token prompt length and using the 600M model, the scoring latency reduces from 33 millisecond to 20 milliseconds when bypassing the decode loop.

Reduce memory synchronization Even after removing the decode loop, expensive per-item work remains. In particular, per-input-token probability calculations can cause significant latency, while such distributions are not used in our ranking setup (only the last token probabilities are useful). Such calculations with additional sampling and eager memory copies delay subsequent GPU kernel launches (large gaps between kernels), leading to increased latency. To this end:

1. We skip internal per-input-token probability extraction when not required.
2. We introduce functions to compute (last token) probabilities without sampling.
3. We optimize memory copy operations between CPU and GPU by replacing many small memory copies with a single vectorized gather on the GPU. We also allow for delaying CPU copy operations so CPU postprocessing overlaps with GPU compute for the next batch.

Taken together, for the 600M model with 300 prompt size, the 99-th percentile latency at 100 RPS (1000 prompts per second) plummeted 92.7% from 6220 milliseconds to 454 milliseconds, allowing for a 25% increase in throughput.

Fixing garbage collection Long-running SGLang servers occasionally experience periodic Garbage Collection (GC) stalls—100–300 milliseconds pauses occurring roughly every 1.5 seconds—caused by Python's generational GC scanning over 750K long-lived objects unnecessarily. These stalls caused us to exceed our p99 latency requirements well before the point of GPU saturation. We fixed this by sending warm up requests to ensure all long-lived data structures are initialized and then invoking Python's gc.freeze() API to exclude all the current objects in memory from future scans.

Amortized prefill optimization Relevance scoring is essentially a batch prefill where all prompts share the same prefix. Therefore, all prompts in a batch share the same prefix Key-Value (KV) cache. Although off-the-shelf SGLang prefix caching can be used here to reuse this KV cache, it requires two forward passes—once with the first prompt and then again with the remaining prompts to leverage the prefix KV cache.

Instead, we perform an "in-batch prefix caching" where we leverage the KV cache from the first item in the batch to compute the attention scores for the remaining items in the batch, all in one forward pass. To this end, we intercept the forward pass between the computation of the KV cache and the attention scores to store the KV cache corresponding to the prefix. Then, for any subsequent prompt with the same prefix, the attention score is obtained via a merge of two attention operations:

1. All suffix tokens attend to all prefix tokens from the first prompt via dense paged attention.
2. All suffix tokens attend to themselves via regular casual attention.

The merge is performed via the Log-Sum-Exp method commonly used to merge two different sets of attention weights.

For large batch sizes, our benchmarks confirm that throughput gains are roughly proportional to the ratio of reduced to

retained token work. Specifically, when the query tokens are skipped, throughput improves according to

$$T = 1 + \frac{N_q}{N_i},$$

where N_q and N_i denote the number of query and item tokens, respectively. For example, with a batch size of 50, $N_q = 50$, and $N_i = 150$, we observe throughput gains of approximately 33%.

5.3 Streaming services optimization

In addition to optimizing the serving inference engine, we perform several optimizations to the streaming and traffic management systems in our semantic search stack, with the goal of increasing the throughput and decreasing GPU under-utilization.

Item score caching In LinkedIn's job search pipeline, a single query can surface hundreds or thousands of potential jobs, yet only 25 results are rendered per page. Each page navigation generates a fresh request to the search service. While the result set changes, the underlying relevance score for a fixed query–item tuple remains deterministic under the same language model. This property makes the scoring step a prime candidate for caching, which not only saves expensive GPU inference cycles but also increases parallel throughput across the ranking service.

To enhance the efficiency of search services, a distributed cache backed by Couchbase was integrated into the system architecture. Prior to invoking the language model, the system performs a cache lookup: cache hits obviate GPU computation, whereas misses trigger model inference followed by cache updates. This design enables repeated navigations or similar queries to exploit previously computed results, thereby reducing computational overhead. Empirical evaluation indicates that with a configurable 15 minutes TTL, over 50% of online scoring requests are satisfied directly from the cache. Consequently, latency reductions were observed across both online and nearline traffic. For online traffic, median latency (p50) decreased by 9.9% and mean latency by 7.3%, with tail latencies improving by 8.3% at p90 and 4.4% at p99. Nearline traffic exhibited similar gains, including an 11.3% reduction at p50, a 9.2% decrease in average latency, and tail latency improvements of 7.5% at p90 and 1.1% at p99. These results substantiate the effectiveness of the caching strategy in mitigating latency, alleviating GPU load, and reducing overall infrastructure costs.

Dynamic scoring depth Search and ranking workloads in large-scale systems rarely follow a flat profile; instead, they oscillate between peak and off-peak periods. To address this variability, we introduced a Dynamic Ranking Depth controller that leverages a PID-based feedback loop to continuously tune how many job candidates flow into the ranking stage. During low traffic, the controller raises ranking depth to exploit unused serving capacity, enabling richer evaluation and higher-quality results for members.

In production, Dynamic Scoring Depth employs a PID controller to adapt ranking depth in response to traffic conditions. Under peak demand, the controller lowers depth just enough to satisfy throughput and latency SLAs while remaining above the neutral-effect threshold to ensure recall is preserved. In controlled product experiments, this adaptive mechanism reduced the average scoring depth from 250 to 131 at peak, yielding a 48% reduction in per-query GPU compute and allowing the system to accommodate substantially higher traffic volumes with a fixed GPU fleet of practical size. Conversely, during off-peak periods the controller increases depth up to 1000, enhancing result quality without compromising responsiveness. By embedding control-theoretic principles into ranking infrastructure, the system dynamically balances relevance, scalability, and responsiveness across highly variable interactive search and alert workloads.

Traffic shaping Traffic shaping techniques have also been introduced to handle latency-insensitive workloads and reduce GPU underutilization. Ranking traffic often exhibits bursty arrival patterns: large waves of queries arrive simultaneously, saturating resources, followed by idle gaps where GPUs sit underloaded. By applying controlled deferral and smoothing requests across these gaps, the system pushes a portion of peak RPS into slightly later windows without violating latency requirements. This evens out GPU scheduling, raises kernel efficiency, and drives higher sustained throughput. Production benchmarks indicate a 25% increase in effective GPU scoring capacity (1600 to 2000 scoring items per second on one Nvidia H100 GPU, measured before and after enabling traffic shaping), demonstrating the impact of shaping burst-heavy workloads through systems-level scheduling.

6 Online Deployment

The final SLM combines model compression, context compression, and SFT to recover performance. To validate our offline results and realize the throughput improvements, we deployed this SLM as the online ranking system in LinkedIn's Semantic Job Search stack (see Section 2), together with the online and streaming optimizations.

Due to the unpruned and uncompressed SLM's inefficiency, we were unable to launch it to a significant volume of LinkedIn users, so a direct online comparison against the unpruned model was infeasible. The majority user experience therefore remained retrieval-based until we could launch the compressed SLM as a second-pass re-ranker. As discussed

Table 9. Online deployment results. The first column shows the performance of the uncompressed SLM against EBR, while the second shows the performance of the compressed SLM against EBR. We report "−" when the results are not statistically significant.

METRIC	SLM V1	SLM V2
POOR MATCH RATE @ 10	−19.84%	−26.69%
NDCG @ 10	+5.54%	+8.45%
JOB SESSIONS	−	+0.59%
JOBS WEEKLY ACTIVE USERS	−	+1.49%
JOB DISMISSES	−	−7.15%

before, we do not deploy any additional re-ranking stages, as we have not observed any business benefits. We present two sets of A/B tests to better understand the optimized SLM's online performance:

1. **Uncompressed SLM (SLM v1) vs EBR:** These results provide a baseline for how a fine-tuned SLM (a 0.5B model) improves ranking quality relative to a retrieval-only search solution that uses an LLM (7B) for embedding-based retrieval.
2. **Compressed SLM (SLM v2) vs EBR:** These results show how the compressed SLM (a 0.6B model reduced to 375M parameters), together with compressed input context, improves upon the same LLM-based EBR baseline.

Note that we updated the SLM's base model from 0.5B to 0.6B in the second test. This produced an offline lift of 2% in NDCG@10, which we traded off for more aggressive compression, yielding offline performance similar to that of SLM v1. Nevertheless, the two SLMs have slightly different architectures: compared with v1, v2 has two additional query heads, six additional KV heads, and four additional transformer layers.

Our results are presented in Table 9. Despite similar offline metrics between SLM v1 and v2, we observe stronger online improvements across the board for v2. The Poor Match Rate metric measures the model's false positive rate. Together with NDCG@10, it indicates that the SLM's relevance has improved (where relevance labels are derived from the SLM's teacher model). Weekly active users and session metrics capture user engagement, while job dismisses indicate dissatisfaction with the returned results. On both fronts, SLM v2 shows statistically significant positive impact. This improved online performance is likely due to architectural changes in v2, including fewer KV heads shared per query head and the presence of additional transformer layers.

In terms of throughput, we achieved an online throughput of 2000 items scored per second per GPU using the 375M pruned model and summarized item descriptions. This is more than a 10× improvement over SLM v1, which did not incorporate the optimizations discussed in this paper. This increase in throughput enabled us to launch LinkedIn's Semantic Job Search to a large segment of users.

Beyond online comparisons with EBR, we compare against our previous production ranking system, LiRank (Borisyuk et al., 2024). LiRank is a DCNv2 (Wang et al., 2021) + TransAct (Xia et al., 2023) based re-ranker operating on top of EBR. Under the same candidate set and evaluation protocol, the SLM re-ranker achieves a 46.7% relative improvement in terms of NDCG@10 over LiRank, marking a significant leap in ranking quality compared to our legacy systems.

7 OPEN SOURCE ARTIFACTS

Many of optimization techniques discussed in this paper can be generalized beyond our semantic search use case. To this end, we have released certain parts of our stack, as we discuss below.

7.1 Model Compression via fmchisel

We have released the PyTorch-based package fmchisel[1] that implement several model compression modules and techniques, replicating much of the pipelines utilized in our production system. In particular, the structured pruning module of fmchisel[2] demonstrate how OSSCAR can be used for pruning a model using any general text dataset (that need not be a semantic search dataset). The distillation module of fmchisel[3] recreates distillation pipelines that enable us to create our base 0.6B model. As an example, we have provided examples in our package for pruning and compressing models for a news excerpt summarization task.

7.2 Efficient inference via SGLang

Many of our improvements to SGLang can be applied to high throughput use cases beyond what we discussed in this paper. For example, SGLang Pull Requests (PRs) #5141 and #9382 implement batch tokenization for high throughput serving scenarios. PR #9241 implements the garbage collection fix to reduce inference latency.

Specialized to scoring use cases beyond LinkedIn's search, PR #9748 implements memory synchronization improvements, resulting in a 25% throughput lift in our use case. PR #8840 allows for skipping the decoding step for any scoring use case. We also encourage the reader to review SGLang

[1] https://github.com/linkedin/FMCHISEL

[2] https://github.com/linkedin/fmchisel/tree/main/examples/structured_pruning

[3] https://github.com/linkedin/fmchisel/tree/main/examples/distillation

issue #15344 where we discuss additional optimizations made to SGLang, beyond the scope of this work.

REFERENCES

Behdin, K., Fatahibaarzi, A., Song, Q., Dai, Y., Gupta, A., Wang, Z., Sang, H., Tang, S., Dexter, G., Zhu, S., et al. Scaling down, serving fast: Compressing and deploying efficient llms for recommendation systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1687–1702, 2025.

Benbaki, R., Chen, W., Meng, X., Hazimeh, H., Ponomareva, N., Zhao, Z., and Mazumder, R. Fast as chita: Neural network pruning with combinatorial optimization. In *International Conference on Machine Learning*, pp. 2031–2049. PMLR, 2023.

Borisyuk, F., Zhou, M., Song, Q., Zhu, S., Tiwana, B., Parameswaran, G., Dangi, S., Hertel, L., Xiao, Q. C., Hou, X., et al. Lirank: Industrial large scale ranking models at linkedin. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4804–4815, 2024.

Chuang, Y.-N., Xing, T., Chang, C.-Y., Liu, Z., Chen, X., and Hu, X. Learning to compress prompt in natural language formats. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7756–7767, 2024.

Dey, S., Braun, B., Ravipati, N., Wu, H., and Li, B. Llmdistill4ads: Using cross-encoders to distill from llm signals for advertiser keyphrase recommendations at ebay. *arXiv preprint arXiv:2508.03628*, 2025.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Gunter, T., Wang, Z., Wang, C., Pang, R., Narayanan, A., Zhang, A., Zhang, B., Chen, C., Chiu, C.-C., Qiu, D., et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Han, E., Chen, J., Sankararaman, K. A., Peng, X., Xu, T., Helenowski, E., Peng, K., Kumar, M., Wang, S., Fang, H., et al. Reinforcement learning from user feedback. *arXiv preprint arXiv:2505.14946*, 2025.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

Hou, Y., Zhang, J., Lin, Z., Lu, H., Xie, R., McAuley, J., and Zhao, W. X. Large language models are zero-shot rankers for recommender systems. In *European Conference on Information Retrieval*, pp. 364–381. Springer, 2024.

Jiang, D. R., Nikulkov, A., Chen, Y.-C., Bai, Y., and Zhu, Z. Improving generative ad text on facebook using reinforcement learning. *arXiv preprint arXiv:2507.21983*, 2025.

Juan, Y., Shen, J., Zhang, S., Shen, Q., Johnson, C., Simon, L., Hong, L., and Zhang, W. Scaling retrieval for web-scale recommenders: Lessons from inverted indexes to embedding search. RecSys, 2025.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *ICLR*, 2024.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Li, Z., Liu, Y., Su, Y., and Collier, N. Prompt compression for large language models: A survey. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7182–7195, 2025.

Liu, Z., Chen, C., Li, W., Qi, P., Pang, T., Du, C., Lee, W. S., and Lin, M. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*.

Meng, C., Arabzadeh, N., Askari, A., Aliannejadi, M., and Rijke, M. d. Query performance prediction using relevance judgments generated by large language models. *ACM Transactions on Information Systems*, 43(4):1–35, 2025.

Meng, X., Ibrahim, S., Behdin, K., Hazimeh, H., Ponomareva, N., and Mazumder, R. Osscar: One-shot structured pruning in vision and language models with combinatorial optimization. In *Forty-first International Conference on Machine Learning*.

Meng, X., Behdin, K., Wang, H., and Mazumder, R. Alps: Improved optimization for highly sparse one-shot pruning for large language models. *Advances in Neural Information Processing Systems*, 37:37594–37625, 2024.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Muralidharan, S., Turuvekere Sreenivas, S., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102, 2024.

Qin, Z., Jagerman, R., Hui, K., Zhuang, H., Wu, J., Yan, L., Shen, J., Liu, T., Liu, J., Metzler, D., et al. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 1504–1518, 2024.

Shang, H., Vo, N., Yadav, N., Zhang, T., Puthenputhussery, A., Cai, X., Chen, S., Chandran, P., and Kang, C. Knowledge distillation for enhancing walmart e-commerce search relevance using large language models. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 449–457, 2025.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Sheng, G., Zhang, C., Ye, Z., Wu, X., Zhang, W., Zhang, R., Peng, Y., Lin, H., and Wu, C. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Shi, X., Liu, J., Liu, Y., Cheng, Q., and Lu, W. Know where to go: Make llm a relevant, responsible, and trustworthy searchers. *Decision Support Systems*, 188:114354, 2025.

Sreenivas, S. T., Muralidharan, S., Joshi, R., Chochowski, M., Mahabaleshwarkar, A. S., Shen, G., Zeng, J., Chen, Z., Suhara, Y., Diao, S., et al. Llm pruning and distillation in practice: The minitron approach. *arXiv preprint arXiv:2408.11796*, 2024.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Wang, H., Sundararaman, M. N., Gungor, O., Xu, Y., Kamath, K., Chalasani, R., Hazra, K. S., and Rao, J. Improving pinterest search relevance using large language models. *arXiv preprint arXiv:2410.17152*, 2024.

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., and Chi, E. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pp. 1785–1797, 2021.

Xia, X., Eksombatchai, P., Pancha, N., Badani, D. D., Wang, P.-W., Gu, N., Joshi, S. V., Farahpour, N., Zhang, Z., and Zhai, A. Transact: Transformer-based realtime user action model for recommendation at pinterest. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5249–5259, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Zhai, J., Liao, L., Liu, X., Wang, Y., Li, R., Cao, X., Gao, L., Gong, Z., Gu, F., He, J., et al. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *International Conference on Machine Learning*, pp. 58484–58509. PMLR, 2024.

Zhang, Y., Li, M., Long, D., Zhang, X., Lin, H., Yang, B., Xie, P., Yang, A., Liu, D., Lin, J., et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Zheng, C., Liu, S., Li, M., Chen, X.-H., Yu, B., Gao, C., Dang, K., Liu, Y., Men, R., Yang, A., et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Zhu, X., Li, J., Liu, Y., Ma, C., and Wang, W. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.