Search

Reimagining LinkedIn's search tech stack

Fedor Borisyuk
January 21, 2026

Co-authors: ***Jiahao Xu***, ***Xiaojing Ma***, ***Sriram Vasudevan***, ***Muchen Wu***, ***Rachel Zheng***, ***Benjamin Le***, ***Shaobo Zhang***, ***Sarang Metkar***, ***Rupesh Gupta***, ***Qianqi Kay Shen***, ***Ali Hooshmand***, ***David Nicolás Racca***, ***Vivek Katarya***, ***Kayhan Behdin***, ***Igor Lapchuk***, ***Xueying Lu***, ***Lingyu(Claire) Zhang***, ***Gokulraj Mohanasundaram***, ***Juan Pablo Bottaro***, ***Lily(Jiayu) Li***, ***Yanbo Li***, ***Guoyao L.***, ***Caleb Johnson***, *and* ***Sundara Raman Ramachandran***

At LinkedIn, our mission is to connect professionals to opportunity. Search plays a central role in this by helping members discover jobs, people, and knowledge that move their careers forward. As the professional landscape evolves, we strive to deliver search experiences that feel relevant, intuitive, personalized, and predictive of what truly matters, from applying to the right job to forming the right connection. That's why we've recently introduced **AI Job Search** and **AI-powered People Search**, which go beyond keyword matching and better understand member intent.

These products reimagine LinkedIn search, using large language models (LLMs) to create a semantic search experience. Instead of relying on exact word overlap between queries and postings, it interprets natural language to infer user goals and preferences. This semantic representation allows for more flexible and accurate retrieval, overcoming vocabulary gaps and aligning search results with how members naturally express their career ambitions.

we'll share how we transformed our overarching search experience at LinkedIn, including the challenges and decisions that went into creating a scalable LLM-based stack and how the technology is powering a smarter, faster, and more personalized experience that helps every member find the most relevant opportunities and connections.

Semantic search's high level infrastructure

When a member submits a query in the search bar, a query understanding module processes the input text, creates a query embedding, and performs embedding-based retrieval (EBR) on CUDA-enabled GPUs using exhaustive vector search (**paper on GPU CUDA-based Search**, **paper on GPU PyTorch-based Search**) to assemble a broad set of candidate documents. The ranking stage then refines these candidates through a Cross-Encoder Small Language Model (SLM) deployed on SGLang, which combines the query, job, and member features to generate relevance and engagement scores for final ranking.

To maintain scalability and efficiency, the ranking pipeline integrates several optimization techniques (**paper on efficient LLM inference infrastructure**, **context compression paper**): score caching, a ranking-depth controller to manage how many candidates progress to deeper ranking, and traffic shaping to balance load during peak times—all designed to enhance latency and result quality. The features and concise job representations consumed by the SLM are produced through a hybrid inference pipeline: a large-scale offline workflow using Spark and Flyte, and a low-latency nearline system using Flink. These embeddings and summaries are stored in distributed storage and retrieved on demand with minimal latency.

In the final stage, the auction layer applies budget and pacing strategies to balance user relevance, engagement, and business metrics, ensuring a healthy

Figure 1. Overall pipeline of LinkedIn's semantic search

Product policy relevance measurement

Measuring relevance quality is essential to delivering a great search experience on LinkedIn. We define product policies that specify how to rate each query–document pair on a five-point scale and use LLM judges to apply these ratings at a massive scale, far beyond what manual evaluation can achieve. These judges are tightly aligned with product managers and engineers through iterative feedback to ensure high agreement. They not only grade tens of millions of query–document pairs daily for relevance measurement but also generate labeled data for training our retrieval and ranking systems, ensuring we optimize search quality according to product policy.

Defining product policy and golden product manager grades

A strong LLM judge begins with a clear product policy and high-quality product manager "golden" grades that demonstrate how that policy should be applied. Product managers act as a "Supreme Court," regularly calibrating to resolve judgment differences and maintain a shared definition of what constitutes a good query–document match. These discussions refine the policy, making it clearer and less subjective. Once product managers reach high agreement (weighted Cohen's Kappa ≥ 0.8), their labels are considered reliable ground truth.

To build a comprehensive golden dataset across diverse user queries, we first categorize queries by attributes (e.g., title–company, name–company, title–skill). Each category is then split into existing user queries and aspirational

sending them to product managers for grading.

Training the LLM judge

Our LLM judge must meet two requirements: high agreement with product managers and the ability to grade tens of millions of query–document pairs daily. To maximize agreement, we collaborate with product managers to prompt-engineer state-of-the-art LLMs, optimizing the weighted Cohen's Kappa Score on golden data. The prompt encodes product-policy guidelines and few-shot examples to drive consistency. While these large models produce high-quality judgments, they cannot meet our throughput needs. To scale, we distill them into a smaller 8B-parameter evaluator LLM. Through supervised fine-tuning on a diverse dataset spanning all query categories and grades, we maintain only small drops in agreement—verified via the Kappa Score on the golden set—while achieving massive efficiency gains.

Continuously measuring quality of search system

Once we have our scalable LLM judge, we can finally build continuous relevance measurement of our system. On a regular basis, we build workflows that perform the following steps:

1. Stratify sample or synthesize a diverse set of queries based on the query categories defined earlier
2. Retrieve the documents returned from executing those queries
3. Decorate documents with additional information required for the correct evaluation/judgement
4. Grade the documents returned using our LLM judge
5. Calculate aggregate precision, recall, and NDCG metrics

Figure 2. Flow of the evaluation process

This workflow serves three primary functions:

- Continuously monitoring the relevance of the overall system
- Evaluating experiments involving underlying ranking and retrieval subsystems
- Distilling student ranking and retrieval models by leveraging evaluation results

Search quality modeling

Search quality is a fundamental requirement for any search product, and achieving it depends on building the core components of a modern search engine. Below, we describe how we leveraged LLMs to enable query understanding, semantic embedding–based retrieval, and cross-encoder ranking.

Embedding-based retrieval

Retrieval is the stage of a search system that identifies a broad set of potentially relevant results from a large corpus. Because no search engine can score every document for every query in real time, we need an efficient retrieval layer to narrow the search space before ranking. Our retrieval sits on top of our GPU-enabled embedding-based retrieval (EBR) system. We built the EBR model by fine-tuning an open-source LLM embedding model to encode queries and jobs into dense vectors. We train on millions of real query–job pairs sampled from production logs, with relevance labels provided by an LLM-based judge. Each query includes its natural language text and query-understanding tags (e.g.

demonstrates a practical path for deploying LLM-based components in real production search systems. The semantic search can directly understand human language as queries, enabling much more intuitive search experiences. This has been a particularly inspiring aspect of the project: bringing modern LLM capabilities into a high-scale, real-time application that serves millions of users.

EBR relevance modeling

To ensure consistency between training and serving, every query is formatted using a lightweight prompt template:

```
Instruct: Given a job search query, retrieve relevant job postings
Query: {query}
{Optional Aspect eg. Company}: {company}
```

The model uses a dual-tower (bi-encoder) architecture: one encoder maps queries to embeddings and the other maps jobs, projecting them into a shared semantic space. Training is end-to-end: we fine-tune all model parameters using Hugging Face Accelerate (with PyTorch FSDP) across multiple GPUs.

We optimize a contrastive InfoNCE loss combined with a margin-based ranking loss. For example, given a query q, a positive job $d+$, and negatives $\{d\text{-}k\}$, we define:

where $sim(\cdot, \cdot)$ is the dot-product similarity of embeddings and τ is a temperature parameter. We also enhance training with hard positives and hard negatives mined from LLM-judged data. Hard positives are LLM-labeled relevant jobs that the current EBR model ranks low, and hard negatives are non-relevant jobs that the model ranks high. These examples reveal exactly where

suppress hard negatives, improving ranking where it matters most:

which encourages *sim(q,d+)* to exceed *sim(q,d-)* by a margin γ. The total loss is a weighted sum, e.g.:

combining the benefits of contrastive and pairwise ranking.

We use multiple evaluation pipelines. First, we leverage production logs for *counterfactual* evaluation: re-ranking the historical candidate list for each query and computing precision, recall and NDCG against true labels. Second, we run offline KNN simulations: we embed held-out queries and job corpus, retrieve nearest neighbors, and directly measure retrieval metrics on this test set. Finally, we integrate the new model into our online serving stack on a subset of traffic to collect end-to-end metrics. These offline metrics provide quick feedback, and the counterfactual log analysis helps estimate the model's user impact without a full live experiment.

Productionization of retrieval

In production, we precompute and store all job embeddings in GPU-backed indexes. At runtime, the incoming query is encoded by the LLM with the prompt aligned with the LLM pre-training to produce its embedding. We then perform an exhaustive k-nearest-neighbor search over the job embedding index using dot-product similarity, returning the top-K jobs. This offline indexing and fast

Query understanding

At the moment a member enters a query into the search bar—the entry point of semantic search—we apply a unified LLM-based understanding layer that interprets intent and converts free text into structured signals. For both AI Job Search and AI-powered People Search, this layer uses fine-tuned 1.5–4B parameter models that meet LinkedIn's latency requirements while delivering high-precision structured outputs (**paper**). A single model handles intent classification, facet extraction, and profile-aware rewriting, replacing multiple brittle NER and heuristic components. The resulting attributes (e.g., title, company, school, location) feed directly into retrieval and ranking.

An intelligent routing layer works alongside this. A lightweight encoder classifies query types at high QPS and performs policy-based safety checks before sending the query down the appropriate path—LLM-powered semantic interpretation for ambiguous inputs or efficient keyword retrieval for precise name and entity lookups.

Together, these components provide a consistent, centrally governed semantic interface for People and Job Search, boosting relevance, simplifying the system, and enabling Semantic Search to scale across LinkedIn's global traffic.

Small language model ranking

The ranking module of semantic search utilizes a Small Language Model (SLM) to estimate how relevant a user's search query q is to each retrieved job_i. The SLM follows a decoder-only architecture. For example, for job search we represent the structured attributes of a job — including its title, company, location, employment type, and remote-work status. Meanwhile people search uses information from the member's profile including their name, company

Here, the *system prefix* and *suffix* contain chat-template tags and explicit instructions guiding the model to determine whether the given job matches the query. When this prompt is passed through the decoder, it produces logits corresponding to the next token. Let $logit_{yes}$ and $logit_{no}$ represent the logits for the tokens “yes” and “no,” respectively. Following prior studies, we compute:

which yields probabilities used to rank job items by their relevance to the user’s query.

SLM training pipeline for relevance quality

Training of the SLM follows a multi-stage process. First, we distill the 7B-parameter teacher model into a compact 0.6B model that can generate graded relevance labels along with rationales. Next, the teacher’s ordinal grades are converted into “soft labels” (p_{yes},p_{no}), representing probabilistic supervision.

We then perform supervised fine-tuning (SFT) to minimize the Kullback–Leibler (KL) divergence between the teacher’s soft targets and the SLM’s predicted probabilities—effectively converting the reasoning-oriented model into a binary relevance classifier:

(LLM). Each pair is evaluated through a structured prompt of the following format:

```
[CRITERIA]: <matching guidelines>
[EXAMPLES]: <reasoning and output format>
[QUERY]: <query text>
[CANDIDATE]: <item text>
Analyze the query-candidate pair and assess how well they align.Provide a single matching score
(0, 1, 2, 3, or 4) along with a brief explanation of your reasoning.
```

The LLM's response includes both a graded relevance score and an accompanying rationale, ensuring that the generated labels are interpretable and consistent with the defined matching criteria.

The model is trained on logged pairs query–job pairs for up to five epochs using the prompt structure. Since job descriptions dominate the input and vary substantially in length (median ≈ 900 tokens; maximum > 2300), we truncate them to ensure the total prompt length does not exceed 2048 tokens during both training and inference. For evaluation, we use a holdout dataset labeled by the teacher model. The SLM ranks job candidates based on p_{yes},p_{no}, and system performance is measured using Normalized Discounted Cumulative Gain at rank 10 (NDCG@10).

Training for multiple objectives

We extend the training paradigm to predict both relevance and engagement within the model, we train a smaller cross-encoder language model (the SLM) using multi-teacher, multi-task distillation. The teachers include:

- The product policy LLM for relevance scoring.

messaging, or following.

For training at scale, we use multiple 1.7B teacher models. To make this feasible, we first distilled our 7B product policy model into a smaller 1.7B version, which serves as a strong but efficient teacher alongside others. During the training process, these teachers run in real time on sampled production query–document pairs, producing soft probability scores that act as supervision targets. The student SLM is trained with KL divergence loss to align its output distribution with the ensemble of teachers. This setup lets us train on real-world data at scale: teacher models provide rich, nuanced probability signals, and the distilled student captures much of their reasoning capacity while remaining lightweight enough to serve millions of queries per second in production.

We have multiple steps of distillation to SLM described in Figure 3, and we show the metrics after distillation in Table 1.

Figure 3. Multi-teacher distillation of SLM

	NDCG@10	Apply AUC	Click AUC
Relevance Teacher 1.7B	0.9484	-	-
Engagement Teacher 1.7B	-	0.8049	0.6772
SLM 0.6B (distilled)	0.9239	0.8007	0.6704

Explainability in search

query. Snippets highlight the most relevant terms and maintain low latency through lazy loading, caching. The approach uses semantic similarity between the query embedding and precomputed phrase embeddings (unigrams/bigrams) from profile text, surfacing the highest-scoring phrases as natural, human-readable snippets. In the offline pipeline, we extract phrases from each profile section (e.g., summary, experience, education), encode them into embeddings in a **Venice key-value store**. This index is refreshed periodically to capture profile updates, with evaluation workflows ensuring quality and readability. At search time, the snippetting midtier receives the query embedding and candidate profiles, fetches stored phrase embeddings, computes cosine similarity, selects top phrases, and expands them into readable snippets using simple heuristics. The final output is a ranked list of profiles paired with snippets and highlighting metadata.

Reasoning

To improve transparency in search results, we introduced a lightweight reasoning module that explains how LinkedIn interprets a member's query. When a query is submitted, the LLM-based understanding model extracts facets, classifies intent, and—using predefined guidelines—produces a concise "thinking state" describing how the query is parsed, along with a brief summary of the types of profiles retrieved. For unsupported or negatively intended queries, the module instead provides a clear explanation of why results may be limited. For example, "berkeley community development specialist msa professional services" becomes "Searching for community development specialist at MSA Professional Services affiliated with Berkeley." To keep latency low, reasoning outputs are cached in Couchbase and reused for repeated or semantically similar queries.

SLM ranking model inference efficiency

- Model pruning, where we remove Fully Connected Layers and remove whole transformer layers.
- Context pruning, by summarization or embedding compression (**paper on AI modeling techniques for efficiency of SLM inference**).

Model pruning

To boost inference throughput, we apply model compression via structured pruning—a technique that removes redundant components to reduce model size and computation with minimal quality loss. Because our models run on GPU infrastructure, we focus on structured pruning, which removes entire neurons, attention heads, or transformer layers so the resulting model can run efficiently on standard GPU kernels. This yields real throughput and latency gains, unlike unstructured pruning, which drops individual weights but often provides no meaningful speedup without specialized hardware.

We prune hidden neurons in Multi-layer Perceptron (MLP) blocks and attention heads in self-attention modules, and we remove full transformer layers to study trade-offs between size, efficiency, and performance (**paper**). Pruning MLP neurons shrinks intermediate activations, while pruning layers reduces network depth—both producing lighter, faster models. After pruning, we fine-tune the model to recover any accuracy loss, ensuring efficiency improvements do not compromise quality.

Context pruning

Item descriptions are long (median ~900 tokens and up to 2,300) making them over 94% of the SLM prompt and causing ~10% of inputs to be truncated at the 2,048-token limit. Removing descriptions severely degrades relevance quality, confirming they carry essential semantic information. To handle this, we use a

descriptions include verbose, often irrelevant details, we train the summarizer with a semantics-preserving loss and a length-aware reward (**paper**). We fine-tune the model using RL to produce concise summaries, balancing (1) reduced input length and (2) preserved model quality. The reward combines:

- a semantic consistency term, measured via KL divergence between the SLM's output distributions on summarized vs. raw text, and
- a length penalty that discourages overly long summaries.

The weighting factor w controls the trade-off between brevity and fidelity, enabling summaries that retain meaning while reducing inference cost.

Embedding compression

Since the computational cost of LLM inference increases quadratically with input length, reducing the number of tokens can greatly decrease overall inference expense. On top of model pruning and summarization, we invented a text–embedding hybrid interaction architecture that condenses each item's text into a single-token embedding generated by an encoder LLM (**paper**). These embeddings are then merged with other textual signals and passed to a ranker LLM for relevance estimation. Because the item embeddings are precomputed and stored in a nearline cache, the volume of text processed during online inference is significantly reduced, resulting in notable improvements in efficiency.

We jointly train the hierarchy of two language models, where one model is producing embedding of the item description and the other model does the ranking of the candidates. On the figure below we show we use two 0.6B models trained jointly. As a result we could replace most of the job description with just a single embedding. We may keep important raw fields with a limited number of

Figure 4. Hierarchy of SLMs trained and served in production

Overall modeling quality and throughput

In the table below we show how quality and inference throughput changed as we improved the modeling technology. As part of the modeling improvement we introduced multitask learning with over 6 tasks of member actions to the model output predictions including relevance, and, for example, were able to improve Click AUC of the model from 0.61 (**baseline**) to 0.67 for Job Ranking.

Setup	NDCG@10	Throughput (ITEMS/SEC/GPU)
SLM with raw-text	0.9432	290
Pruned SLM and SUMMARIZED TEXT	0.9218	2200
SLM with EMBEDDING COMPRESSION	0.9239	22000
Embedding based retrieval (EBR) baseline	0.838	>1.6B, exhaustive search on GPU

Looking ahead

We built a modern semantic search system by first establishing an LLM-based quality evaluation framework grounded in product policy. On top of this foundation, we introduced LLM-powered query understanding, semantic retrieval, and ranking models—driving double-digit improvements in search

running in production. As we continue refining Semantic Search, our focus remains on empowering every member to discover the right opportunity, connection, or insight—at the right time.

Acknowledgements

To the more than 100 team members who've contributed to AI-powered job search and people search across infrastructure, AI modeling, user experience, and data science: thank you for your creativity, grit, and teamwork. This milestone belongs to all of us.

Related articles