SRCB at TREC 2025: Million LLMs and Tip-of-the-Tongue Tracks

Hongyu Li, Yuming Zhang, Junyu Zhou, Yongwei Zhang, Shanshan Jiang, Bin Dong
Ricoh Software Research Center (Beijing) Co., Ltd
{Hongyu.Li, Yuming.Zhang1, Junyu.Zhou, Yongwei.Zhang, Shanshan.Jiang, Bin.Dong}@cn.ricoh.com

Abstract

This paper reports the performance of SRCB's system in the Million LLMs and Tip-of-the-Tongue tracks. For the Million LLM Track, we mainly rely on powerful LLMs and various methods to construct the missing training labels. And then, we use the constructed training data and devise several approaches to achieve the ranking target and conduct experiments. For the Tip-of-the-Tongue task, we propose a retrieval framework that integrates dense and LLM-based components. Original queries are transformed into cue lists, and additional data are used to fine-tune both the dense retriever and re-ranker. Furthermore, retrieval results from LLMs are incorporated to supplement the reranker, and the final ranking is produced using an LLM reranker.

1 Introduction

In recent years, information retrieval has become more challenging due to the growth of large language models (LLMs) and the need to help users find things they partially remember. The TREC2025 Million LLM Track and Tip-of-the-Tongue (ToT) Track focus on these two important areas, each with its own challenges.

The Million LLM Track[1] asks systems to predict which LLM is most likely to give a correct answer for a given query. Instead of retrieving documents, systems rank LLMs based on precomputed responses and metadata. This track tests the ability to estimate expertise and choose the best model for each question.

The ToT Track[2] (Arguello et al., 2023) focuses on cases where a user remembers only parts of what they are looking for, such as a movie, a person, or a place. The task is to find the correct item in a large corpus, like Wikipedia, based on incomplete or vague descriptions. This requires systems to handle fuzzy information, partial hints, and uncertainty in user memory.

For the Million LLM Track, where ground-truth labels are unavailable, we employ methods based on powerful LLMs to construct training labels. These methods primarily leverage the intrinsic capabilities of such LLMs to act a proxy for human assessment in evaluating how well a response answers the query. The resulting labels are subsequently used for model fine-tuning. Morerover, to accommodate the scale of the training data, we mainly adopt the "Pretrained Model + Decoder" architecture, in which the parameters of the pretrained model remain frozen during the fine-tuning process. Additionally, we design multiple ranking approaches to systematically rank all 1,131 candidate responses for each query.

For the ToT Track, we present a retrieval framework combining dense and LLM-based components. A dense retriever first retrieves the top 2,000 candidates, which are then refined by a re-ranker, with both stages built on the Qwen3-Embedding series (Zhang et al., 2025a). While this backbone achieves strong performance in celebrity and landmark domains, the movie domain exhibits greater room for improvement; to address this, we enhance the training data with additional movie-domain instances and construct three variants to fine-tune both the retriever and re-ranker. Building on this pipeline, we further incorporate LLM-based components: an LLM retriever leverages the model's inherent knowledge to retrieve candidates across all domains, and its outputs are merged with those of the conventional re-ranker, followed by an LLM re-ranker that produces the final ranking, resulting in improved retrieval and ranking accuracy across diverse domains.

[1] https://trec-mllm.github.io/
[2] https://trec-tot.github.io/

2 Million LLMs Track

This track addresses the task of ranking the likelihood that numerous large language models (LLMs) generate high-quality responses to a given query—prior to actual response generation. The core objective lies in learning the knowledge and distribution from the training dataset about which LLMs are more suitable for responding to specific types of queries. The training dataset comprises 1,131 candidate LLMs and nearly 15,000 queries. And each LLM has a response for each query, resulting in millions of query-response pairs. In addition, there are no ground-truth labels in the training dataset to indicate the relative ranking of these LLMs. The two main challenges of this task are (1) the lack of ground-truth ranking labels and (2) the large scale of the dataset.

2.1 Labels Construction

To overcome the challenge of the lack of ground-truth ranking labels, we explore and leverage our system to automatically annotate the query-response pairs in the training dataset using the following distinct approaches.

2.1.1 Quicksort

We adopt a Quicksort-inspired pairwise comparison approach using a powerful LLM to evaluate all 1,131 responses per query. Specifically, for each query, we randomly select one response from 1,131 LLM responses as the anchor response and compare all other responses with it. The powerful LLM is then prompted to judge whether a given response is better than the anchor response. Responses judged as better are assigned to a 'good' list, while those judged worse are placed in a 'bad' list. The responses are thereby partitioned, establishing the ranking: 'good' list > the anchor response > 'bad' list. This process is applied recursively to all sub-lists (both 'good' and 'bad' lists) until a full ranking is obtained. We use these full ranking labels generated by the powerful LLM to fine-tune our models. Moreover, in practice, we also account for ties—instances where responses are literal identical or judged equally good—by assigning them the same score as the anchor response.

2.1.2 Re-ranker

We employ a powerful re-ranker model (Qwen3-Reranker-8B, (Zhang et al., 2025b)) to evaluate the quality of each LLM response. Each query–response pair is formatted as: "whether the response can be the correct reply to the query?" We collect the likelihood of the "Yes" token from the logits output of the model and we treat it as the quality score of the response. All 1,131 responses for a query are then ranked based on this score.

2.1.3 Embedding Similarity

Standard Response: We first use a powerful LLM to generate a standard (reference) response for each query, assuming that it represents a perfect answer. We then compute the similarity between each candidate response and this standard response using a powerful embedding model. The similarity score is used to rank all candidate responses.

Query-based Method: Alternatively, we directly compute the embedding similarity between each candidate response and the original query and use this score for ranking.

2.2 Approach

Given the scale of the dataset, we adopt an efficient fine-tuning strategy by freezing the pretrained model parameters and optimizing only the task-specific decoder components. This significantly reduces memory usage. We propose three main learning paradigms: **Ranking**, **Classification**, and **Regression**.

2.2.1 Ranking

In this approach, we treat the task objective as a direct ranking task over all LLMs. We append a fully connected layer on top of the frozen encoder, with an output dimension of 1,131—each output node corresponds to the predicted likelihood of one LLM producing a high-quality response. The LLMs are then ranked based on these scores.

2.2.2 Classification

In this approach, each LLM is treated independently. For each of the 1,131 LLMs, we attach a separate classification head that outputs a 3-class distribution: (a) highly related, (b) related, or (c) unrelated. This setup aligns with the label schema in the development dataset.

Since our system constructed labels are continuous scores, we map them into three categories by setting thresholds. We experimented with various percentile splits to assign the top x% as "highly related", the next y% as "related", and the remainder as "unrelated." These thresholds were treated as hyperparameters during optimization.

2.2.3 Regression

This approach is similar to **Classification**, but it reformulates the classification task as a regression task. For each of the 1,131 LLMs, we attach a separate regression head using a Sigmoid activation function, which outputs a continuous value between 0 and 1. We observe that even the regression method also requires the top x% pre-processing step. When labels outside the top x% are set to 0—rather instead of retaining their original decimal values—the model exhibits improved performance.

2.3 Experiments

We evaluate results using three different random seeds to ensure robustness. We design five baseline methods by combining the two learning paradigms with different loss functions:

- **Ranking** with (a) Binary Cross Entropy Loss (BCE) and (b) List Net Loss (LN) (Cao et al., 2007)
- **Classification** with (a) Cross Entropy Loss (CE) and (b) Focal Loss (Focal) (Lin et al., 2017)
- **Regression** with Binary Cross Entropy Loss (BCE)

To empirically investigate the influence of key design factors, we perform an extensive set of experiments aimed at quantifying their individual and collective contributions to the final outcome. The following experiments present analyzes of different design configurations through a comprehensive set of experiments, assessing their impact on performance.

Effect of LLM Size in Quicksort As shown in Table 1, the labels generated by Qwen3-14B consistently produce the best performance across all sizes.

	0.6B	14B	32B
Ranking + BCE	38.7%	**42.2%**	37.6%
Ranking + LN	38.1%	**41.8%**	37.6%
Classification + CE	34.5%	**41.9%**	38.5%
Classification + Focal	32.1%	**40.5%**	38.8%
Regression + BCE	35.8%	**42.1%**	38.5%

Table 1: Performance with different LLM sizes used in Quicksort label construction. All LLMs are from the Qwen3 series.

Comparison of Label Construction Methods Table 2 shows that Quicksort and the combined method perform the best among all label construction strategies.

Choice of Pretrained Language Model Table 3 indicates that LLM-based encoders (Qwen3-Embedding-8B and Qwen3-32B) outperform DeBERTa-v3-base (He et al., 2021), likely due to their larger capacity. Notably, Qwen3-Embedding-8B achieves competitive or better results than Qwen3-32B, possibly due to its specialized training for embedding tasks.

2.4 Submissions

In our subsequent experiments, we mainly focus on the label construction methods of Quicksort and Combination, and optimize all five learning paradigms as well as optimize in the form of Qwen3-Embedding-8B as the encoder. During the optimization process, we adjust the hyperparameters like learning rate, batch size, dropout, and some specific parameters related to certain methods, such as the top x "highly related" as mentioned before. During the experiments, we collect some of the best-performing models, which are used to make predictions on the test set. Then, the final submission results are obtained through the majority voting method.

3 Tip-of-the-Tongue Track

3.1 Approach

Our system follows a standard pipeline consisting of a dense retriever and a re-ranker. The dense retriever first retrieves the top 2,000 candidates, which are then passed to the re-ranker for further ordering. We use the Qwen3-Embedding series as our backbone models. We observed that Qwen3-Embedding performs very well on the celebrity and landmark domains in dev3, while the movie domain shows greater room for improvement. To address this gap, we incorporated additional movie-domain data and constructed three variants of augmented data to fine-tune both the dense retriever and the re-ranker, aiming to improve retrieval and re-ranking accuracy.

Building on this pipeline, we further introduce LLM-based components, including an LLM retriever and an LLM re-ranker. The LLM retriever directly leverages the model's inherent knowledge to retrieve candidates for queries across all domains. Its results are merged with those of the re-ranker. Finally, an LLM re-ranker produces the final ranking.

	Quicksort	Re-ranker	ES (Standard Response)	ES (Query)	Combination
Ranking + BCE	42.2%	38.8%	40.8%	37.5%	**42.3%**
Ranking + LN	41.8%	38.1%	41.0%	37.5%	**42.1%**
Classification + CE	**41.9%**	39.1%	40.6%	37.8%	41.5%
Classification + Focal	40.5%	37.5%	39.7%	36.7%	**40.8%**
Regression + BCE	42.1%	39.4%	41.2%	38.3%	**42.2%**

Table 2: Comparison of Label Construction Methods. ES: Embedding Similarity. Combination: ensemble of multiple methods

	Deberta-v3-base	Qwen3-Embedding-8B	Qwen3-32B
Quicksort			
Ranking + BCE	42.2%	**43.8%**	43.7%
Ranking + LN	41.8%	**43.9%**	43.5%
Classification + CE	41.9%	**43.5%**	43.3%
Classification + Focal	40.5%	42.8%	**42.9%**
Regression + BCE	42.1%	**46.5%**	45.7%
Combination			
Ranking + BCE	42.3%	**45.2%**	45.2%
Ranking + LN	42.1%	**45.7%**	45.7%
Classification + CE	41.5%	**45.2%**	44.7%
Classification + Focal	40.8%	**44.7%**	44.6%
Regression + BCE	42.2%	**45.8%**	45.2%

Table 3: Choice of Pretrained Language Model

3.1.1 Retriever

The retriever module applies four techniques to improve retrieval performance: (1) query preprocessing to transform raw queries into cue lists, (2) a dense retriever based on Qwen3-Retriever-8B[3], fine-tuned on augmented data with three variant types, (3) candidate filtering using Wikidata to retain only entities relevant to the movie domain, and (4) LLM-based retrieval to complement the dense retriever results on the test set.

Query preprocessing Consistent with the observations reported in Adhikary et al. (2024); Fröbe et al. (2024), we observe that the raw ToT queries in the dataset often fail to provide clear or effective cues for retrieval. To address this, we use DeepSeek-V3 (Liu et al., 2024) to transform each original query into a refined list of cues. Empirically, this transformation removes redundant or noisy content, highlights information that is most useful for retrieval, and corrects typos and grammatical errors. Throughout this paper, all modules take cue lists as input rather than raw queries. See the prompt template in Table 4.

[3] https://huggingface.co/Qwen/Qwen3-Embedding-8B

Candidate filtering with Wikidata Our goal is to improve recall in the movie domain by reducing the number of irrelevant Wikipedia article candidates. To achieve this, we leverage Wikidata to assess the domain relevance of each entity. Specifically, we use the "instance of" properties from the Wikidata dump to estimate whether a given entity is related to movies. For each entity, we compute the cosine similarity between the embedding of every "instance of" value and the embedding of "Movie". All embeddings are obtained using the Qwen3-Embedding-8B model. The highest similarity score among these properties is then taken as the entity's relevance score to the movie domain. We then filter out entities whose relevance scores fall below a predefined threshold, and remove their corresponding Wikipedia articles via URL alignment. Using this procedure, we reduce the original set of 6.4 million candidate Wikipedia articles to approximately 0.5 million articles that are relevant to the movie domain.

Dense retriever and fine-tuning We use Qwen3-Retriever-8B as the backbone of our dense retriever and build the index for all Wikipedia articles using the first 512 tokens of each. To further improve the performance of the dense retriever on the movie

Query Preprocessing Prompt Template

You are an intelligent assistant helping users with Tip-of-the-Tongue (ToT) known-item retrieval—situations where someone is trying to recall a specific entity (e.g., a movie, book, person, song, place, event, etc.) they previously encountered but can't remember a reliable identifier like the name or title. Instead, they provide vague or partial memories. Your task is to extract a list of cues from the user's input. Cues are any meaningful pieces of information that may help in identifying the intended entity.

Cues may include (but are not limited to):
1. Descriptions of appearance, content, or notable features
2. Plot fragments, scenes, events, or quotes
3. Sensory details (visuals, sounds, smells, etc.)
4. Emotions or atmosphere
5. Context of experience (e.g., "saw it as a kid," "on a trip to Italy," "heard it on the radio")
6. Time period when experienced or when it was popular/released
7. Language, region, or culture of origin
8. Associated people, groups, or objects
9. Comparisons (e.g., "felt like a mix between X and Y")
10. Any other detail, however imprecise, that the user remembers

Return the cues in a clear, structured list. Be concise but retain nuance. Directly output the list, and do not attempt to guess the actual entity—just extract the cues.

{*examples*}

Input: {*cue list*}

Table 4: Prompt used in query preprocessing.

domain, we use the training set, the dev1 set, and 1,000 samples drawn from the Webis' Tip-of-My-Tongue Known-Item Search Triplets dataset[4] as our training data and fine-tune the Qwen3-Embedding-8B model. Since ToT information requests are often vague, uncertain, or may contain errors and misleading cues, we further construct three augmented variants for each training sample to enhance model robustness. Specifically, we apply the following operations:

1. **Cue deletion**: remove one "unimportant" cue from the cue list. The importance score is defined as the cosine similarity between each cue and the gold document.

2. **Synonym substitution**: replace one cue with an appropriate synonym.

3. **Misleading substitution**: replace one cue with an alternative phrase that intentionally alters the meaning and changes the search intent.

[4] https://huggingface.co/datasets/webis/tip-of-my-tongue-known-item-search-triplets

Based on the fine-tuned embedding model, we build an index for the articles relevant to the movie domain.

LLM Retrieval Modern powerful LLMs are typically pretrained on corpora that include Wikipedia, making them capable of directly recalling Wikipedia article titles and URLs from textual queries. Leveraging this property, we prompt LLMs to retrieve relevant Wikipedia entries based on each input query. Specifically, we take the union of results returned by DeepSeek-R1 and GPT-5, and keep the top 10 candidates. We then align the retrieved titles and URLs—either directly via the URL provided by the model or by querying the Wikimedia API—ensuring that each recalled item corresponds to an article in our Wikipedia corpus. Due to budget constraints, this LLM-based recall strategy is applied only to the test set and is not evaluated on the development sets.

3.1.2 Re-ranker

The re-ranker module is designed as a multi-stage refinement system that enhances ranking quality under vague and noisy ToT queries. It consists of

three key techniques: (1) Fine-tuning a pointwise Qwen3-8B-Reranker[5] model with carefully controlled negative sampling to enhance discrimination, (2) Applying a LLM-based listwise reranking step to refine the top-K candidates, and (3) leveraging synonym substitution and Ranking ensemble to improve robustness.

Pointwise reranking We adopt Qwen3-8B-Reranker as our reranking backbone and apply LoRA to fine-tune all linear layers. We train two pointwise rerankers: one specialized for movie queries and the other one for general queries across all categories.

For the movie reranker, the training set includes *train queries*, *dev1 queries*, *dev2 queries*, and 300 triplets from the public *Webis Tip-of-My-Tongue Known-Item Search Triplets* dataset (Fröbe et al., 2024). For the general reranker, the training data consist of the above four sets and synthetic examples generated using GLM-4-plus (GLM et al., 2024). The *dev3* dataset is reserved exclusively for evaluation of both rerankers.

The selection of negative samples plays a critical role in training an effective reranker. We begin by using Qwen3-8B-Reranker to rerank the top-2000 candidates and extract negative samples from this baseline ranking. We categorize negative samples into three difficulty levels—hard, medium, and easy—corresponding to rank ranges 1–10, 10–100, and 100–2000, respectively. Extensive experiments show that a negative sampling ratio of $1:3:3$ across these three levels yields the most stable and competitive reranking performance. In other words, after data augmentation, each query–document pair is expanded such that the ratio of positive to negative samples becomes $4:28$.

For reranker training, we use a unified prompt template. The reranker is required to judge whether a given document satisfies the information need expressed in the query under the provided instruction. The model must output strictly "yes" or "no". The full prompt template is shown in 5, in which the document has been truncated to less than 1024 tokens.

Listwise Reranking During our experiments, we observe that the reranker model often fails to surface the correct answer because the relevant Wikipedia articles do not contain the key information implied by the vague memory cues. To mitigate this limitation, we take 10 additional candidates from the LLM retrieval results described in 3.1.1 and insert them to the reranking results.

To further improve the NDCG performance, we apply an additional listwise reranking stage on the top-20 and top-10 candidates produced by our reranker. Empirically, we observe that for queries whose correct target entity already appears within the top-20 results, the correct label is typically ranked within the top-5 positions. This motivates us to introduce additional LLM-generated recommendation candidates while carefully controlling their insertion position to avoid pushing the true label too far down in the ranking.

Specifically, we experiment with inserting LLM-recommended candidates at different cutoff points—such as after the 6th or 10th position of the reranker's output. This strategy allows us to enrich the candidate set while preserving the high-ranking position of the true label, thereby minimizing negative impact on NDCG.

For listwise reranking, we truncate each candidate Wikipedia passage to the first 512 tokens and prepend the entity name to form the final ranking input. We then query the DeepSeek-V3 (Liu et al., 2024) using a prompt template adapted from RankGPT (Sun et al., 2023), instructing the model to listwise rank the passages based on their relevance to the cue list. The prompt template is shown in 6. This two-stage reranking pipeline—pointwise reranker followed by LLM-based listwise reranking—provides complementary strengths and yields consistent improvements in NDCG.

Pointwise Finetuning Prompt Template

<|system|>
Judge whether the Document meets the requirements based on the Query and the Instruct provided. Note that the answer can only be 'yes' or 'no'.

<|user|>
<Instruct>: Given some cue lists extracted from the tip of the tongue memory, retrieve relevant passages that answer the query.
<Query>: {*cue list*}
<Document>: {*the truncated document*}

Table 5: Prompt used in pointwise fine-tuning.

[5] https://huggingface.co/Qwen/Qwen3-Reranker-8B

Listwise Reranking Prompt Template

<|system|>
You are RankGPT, an intelligent assistant that can rank wikipedia entities based on their relevancy to the query.

<|user|>
I will provide you with {K} entities, each indicated by number identifier []. Rank the entity based on their relevance to cue list extracted from tip of the tongue memory: {*cue list*}.

<|assistant|>
Okay, please provide the wikipedia entities.

<|user|>
[1] {*Document 1*}
<|assistant|>
Received passage [1].
...
<|user|>
[K]{*Document K*}
<|assistant|>
Received passage [K].

<|user|>
Tip of the tongue memory: {*cue list*}.
Rank the {K} wikipedia entities above based on their relevance to the ToT memory. The entity should be listed in descending order using identifiers. The most relevant entity should be listed first. The output format should be [] > [], e.g., [1] > [2]. Only response the ranking results, do not say any word or explain.

Table 6: Prompt used in listwise reranking.

Ranking ensemble We enhance robustness on ambiguous ToT queries by applying a ranking ensemble strategy based on Reciprocal Rank Fusion (RRF) (Cormack et al., 2009). RRF is is an ensemble method that combines multiple ranked lists by assigning each item a score based on the reciprocal of its rank in each list. It boosts items that consistently appear near the top across different rankers, improving overall retrieval robustness.

For each test query, three synonym-based paraphrases are generated using an LLM, and all four versions of the query (the original plus three reformulations) are processed through the full retrieval and reranking pipeline. This yields four ranked lists with complementary relevance signals, which are then ensembled with RRF to produce a more stable and reliable final ranking.

3.2 Submissions

We submitted four different rankings as the final submission, each implementing a multi-stage retrieval and reranking pipeline. The details of the four submissions are as follows:

- **srcb-tot-01:** A pipeline composed of a Dense Retriever, a Reranker, and an LLM Reranker. Queries are first converted to cues using *DeepSeek-V3*. In the movie domain, we use a finetuned *Qwen3-Embedding-8B* model; other domains use the base embedding model. The Reranker is a finetuned *Qwen3-Reranker-8B*. We rerank the top 2000 retrieved candidates, and *DeepSeek-V3* performs listwise reranking of the top 20.
- **srcb-tot-02:** A pipeline consisting of a Dense Retriever, Reranker, LLM Retriever, and LLM Reranker. The initial retrieval is the same with srcb-tot-01. *DeepSeek-R1* retrieves up to 10 Wikipedia entities, after which the candidates ranked 11–20 are replaced by LLM retrieval results. The top 20 candidates are then reranked by the LLM Reranker.
- **srcb-tot-03:** This pipeline uses a Dense Retriever, Reranker, LLM Retriever, and LLM

Run	NDCG@10	NDCG@1000	MRR@1000	R@1000
srcb-tot-01	0.6162	0.6458	0.5844	0.8955
srcb-tot-02	0.6449	0.6700	0.6097	0.9051
srcb-tot-03	0.6547	0.6787	0.6211	0.9051
srcb-tot-04	**0.6576**	**0.6824**	**0.6258**	**0.9051**

Table 7: Results of the four runs submitted to TREC.

Reranker with a three-stage ranking strategy. LLM retrieval results are inserted at rank 6, *DeepSeek-V3* listwise reranks positions 2–10, and *GPT-5* performs fine-grained reranking of the top 4 candidates.

- **srcb-tot-04:** Similar to srcb-tot-03, this pipeline combines Dense Retriever, Reranker, LLM Retriever, and LLM Reranker, but applies different reranker finetuning for movie and non-movie domains. For the movie domain, it follows the three-stage pipeline as in srcb-tot-03; for other domains, candidates ranked 11–20 are replaced with LLM retrieval results and then reranked top 20.

The evaluation results of the submitted runs are shown in Table 7

References

Subinay Adhikary, Shuvam Banerji Seal, Soumyadeep Sar, and Dwaipayan Roy. 2024. Iiserk@ tot_2024: Query reformulation and layered retrieval for tip-of-tongue items. In *33th International Text Retrieval Conference (TREC 2024)(NIST Special Publication), Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST)*.

Jaime Arguello, Samarth Bhargav, Fernando Diaz, Evangelos Kanoulas, and Bhaskar Mitra. 2023. Overview of the trec 2023 tip-of-the-tongue track. In *The Thirty-Second Text REtrieval Conference Proceedings (TREC 2023), Gaithersburg, MD, USA, November*, pages 14–17.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*.

Maik Fröbe, Lukas Gienapp, Jan Heinrich Merker, Harrisen Scells, Eric Oliver Schmidt, Matti Wiegmann, Martin Potthast, and Matthias Hagen. 2024. Webis at trec 2024: Biomedical generative retrieval, retrieval-augmented generation, and tip-of-the-tongue tracks. In *33th International Text Retrieval Conference (TREC 2024)(NIST Special Publication), Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST)*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, and 37 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

W Sun, L Yan, X Ma, P Ren, D Yin, Z Ren, H Bouamor, J Pino, and K Bali. 2023. Is chatgpt good at search?: Investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025a. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025b. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.