How to Choose the Right LLM?
Exploring Methods for the Million LLMs Track

Catalina Riano and Hui Fang

University of Delaware, USA

December, 2025

1 Introduction

The field of Information Retrieval (IR) is witnessing a fundamental shift; users now interact not only with traditional search engines, but also with AI interfaces where Large Language Models (LLMs) provide synthesized conversational answers [1]. In this answer-centric paradigm, the challenge moves from retrieving documents to determining which model is best suited to address a given query, especially as modern solutions contain vast ensembles of different types of LLMs with diverse reasoning skills, domain knowledge, linguistic capabilities, and response styles. Some models excel at detailed explanations, others at concise factual outputs; some are for general purposes, while others specialize in specific domains [2]. As this diversity continues to expand, selecting the most appropriate model for each query becomes a central problem in this area of study.

The Million LLMs Track, part of TREC 2025, addresses the challenge of determining which LLMs are best suited to answer a specific user query. The main goal of the track is to evaluate the ability of a system to predict LLM expertise: given a query and a set of LLM IDs, the system must rank the models by how likely they are to provide a high quality answer. Unlike traditional IR settings, this ranking must be produced without querying the models at test time. Instead, participants must rely on precomputed discovery data, including LLM responses, metadata, and development labels, to infer each model's strengths and capabilities. For every query, the required output is a ranked list of LLMs, ordered from most to least likely to generate a good answer, following the standard TREC run format [3].

This report explains our submission to the Million LLMs Track and outlines the methods we implemented for the task. Our approaches primarily adapt known established techniques, tailoring them to the targets of the track and the characteristics of the provided data. In this document, we describe the different strategies implemented and their details, while also providing context on the formulation, datasets, and evaluation of the task. Although official results are not yet available, the methods explored here aim to further the understanding of the techniques capable of selecting the most appropriate LLMs for a given query.

2 Million LLMs Track: Summary and Details

The goal of the Million LLMs track is to evaluate a system's ability to predict which LLMs are most capable of answering a given query. Formally, given a user query and a set of LLM identifiers, the system must rank the models by predicted expertise, estimating how likely is each model to produce a high quality answer without issuing new queries at test time. For each query, the required submission is a ranked list of LLM IDs, ordered from most to least likely to deliver a good response [3].

The datasets supporting this task are:

- *Discovery Set:* contains 14,940 training queries, the precomputed responses from 1,131 LLMs for each query, and the associated metadata.

- *Development Set:* contains a set of 342 evaluation queries paired with graded expertise score labels.
- *Test Set:* contains 50 queries for the final assessment and submission to the track.

The performance of the submitted methods, or runs, will be assessed using relevance ranking metrics. The first metric used is the Normalized Discounted Cumulative Gain in the top 10 ranked results (nDCG@10), which evaluates ranking quality by computing discounted cumulative gain over graded relevance labels, emphasizing the correct placement of highly expert models near the top of the ranking. The second metric used is the Mean Reciprocal Rank (MRR), which captures how early the first highly relevant LLM appears on the ranked list. All evaluations are performed against hidden ground-truth expertise labels. At the time of this document submission, official results are not available. However, in this report, we present the results of the performance of each method on the Development Set.

3 Method 1: Hierarchical LLM Profiles

Method 1 is a hierarchical, two-stage approach that combines coarse global information about each LLM with fine-grained evidence derived from individual answers, building an expertise profile for each model using only its historical discovery answers. At a high level, the method assumes that an LLM's expertise can be inferred from the semantic space of its past responses: if a new query is close to regions of this space where a model has frequently produced good answers, that model is likely to perform well again. The method is fully automatic and internal, meaning that all profiles and scores are learned exclusively from the Discovery Set, and no manual intervention is performed after seeing the test queries.

The first stage, *Cluster Profiles (Broad Scoring)*, builds a global representation of each LLM's expertise. All answers in the Discovery Set are embedded into a shared semantic space using a sentence embedding model, and for every LLM m, these embeddings are grouped into clusters whose centroids represent distinct expertise regions. At inference time, a new query is embedded and normalized to obtain q. For each LLM m, let $\{c_{m,1}, \ldots, c_{m,K_m}\}$ denote its cluster centroids. The *cluster-based (broad) score* measures how well the query aligns with the model's global profile and is defined as

$$s_{\text{broad}}(q, m) \;=\; \max_{1 \le k \le K_m} \langle q,\, c_{m,k} \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the cosine similarity (implemented as a dot product between normalized vectors). Intuitively, if the query embedding lies close to one of the dense regions where model m has frequently produced answers, then $s_{\text{broad}}(q, m)$ will be high. All LLMs are scored in this way, and only the top M candidates according to $s_{\text{broad}}(q, m)$ are retained. This step acts as a global, profile level filter that discards clearly mismatched models before the second stage refinement.

The second stage, *Answer Embedding Index (Refined Scoring)*, provides a more localized signal. For each LLM, all of its answers are embedded and stored in a vector index that supports efficient k-nearest-neighbor (kNN) search [4]. At inference time, the query embedding q is used to retrieve the top-K most similar answer embeddings $\{x_{m,1}, \ldots, x_{m,K}\}$ from model m's index, with corresponding similarity scores $\{d_{m,1}, \ldots, d_{m,K}\}$. These similarities are aggregated into a refined score

$$s_{\text{refine}}(q, m) \;=\; \frac{1}{K} \sum_{j=1}^{K} d_{m,j},$$

which reflects how often model m has produced answers that are very similar to what the current query appears to require. This refined stage captures fine-grained expertise that may not be fully represented by cluster level scoring alone.

The final prediction for a given query and LLM is obtained by a weighted combination of the broad and refined scores. For each query q and LLM m, Method 1 computes a broad cluster-based score $s_{\text{broad}}(q, m)$ and a refined nearest-neighbor score $s_{\text{refine}}(q, m)$. These are combined into a final score

$$s_{\text{final}}(q, m) \;=\; w \cdot s_{\text{refine}}(q, m) \;+\; (1 - w) \cdot s_{\text{broad}}(q, m),$$

where $w \in [0, 1]$ controls the trade-off between global profiles and local evidence. The broad cluster based score encodes global expertise patterns across large regions of the embedding space, while the refined answer based score focuses on local neighborhoods of semantically similar examples. By interpolating these two signals, the method balances robustness (from broad profiles) with sensitivity to query specific details (from nearest neighbor retrieval). For each query, LLMs are then ranked in descending order of $s_{\text{final}}(q, m)$, yielding the final submission ranking.

Implementation Details Method 1 uses only the provided Discovery Set data. All queries and answers are embedded using the `BAAI/bge-large-en` embedding model [5], and the resulting vectors are ℓ_2-normalized so that cosine similarity can be implemented as a dot product.

For the broad component, we apply standard k-means clustering independently to the normalized answer embeddings of each LLM, using a fixed number of clusters per model, $K_m = 10$. The resulting centroids define a compact cluster profile that summarizes the regions of the embedding space where each LLM tends to produce answers.

For the refined component, we build a separate FAISS index [6] over the normalized answer embeddings for each LLM and, at inference time, retrieve the top-$K = 10$ nearest neighbors of the query embedding from each index to estimate local expertise. The final score for each (query, LLM) pair is obtained by interpolating the refined, neighbor based score with the broad, cluster based score using a mixing parameter $\alpha = 0.2$, which controls the trade-off between local evidence and global profile information. All hyperparameters ($K_m = 10$, $K = 10$, $\alpha = 0.2$) were chosen based on Development Set performance and kept fixed for the reported runs.

4 Method 2: Hybrid Ranking Model

Method 2 is a hybrid approach that treats LLM selection as a problem of estimating expertise directly in the *query* space, guided by weak supervision from an external LLM judge [7]. The core assumption is that similar queries should induce similar expertise patterns: if two queries are close in the embedding space and a given model performs well on one, it is likely to perform well on the other. Unlike Method 1, which operates primarily in the answer space, Method 2 builds its profiles over query embeddings and uses weak labels to derive both local and domain level expertise signals. The method is automatic and external: it uses only the provided Discovery Set plus an external LLM as a judge, with no manual tuning after seeing the test queries.

The first part is the weak labeling. For every discovery query q_d and LLM m, the corresponding answer is evaluated by an external LLM, which assigns a graded quality score $g(q_d, m)$. These scores are not used directly at test time, but they define how good each LLM is on each training query. All discovery queries are embedded using a sentence encoder, and the resulting vectors are stored in two structures: (i) a k-nearest-neighbor (kNN) index over query embeddings, and (ii) a clustering of queries into a fixed number of expertise regions. At inference time, given a new query q, the *local* stage focuses on query specific evidence. We retrieve the K nearest discovery queries $\{q_{d1}, \dots, q_{dK}\}$ from the kNN index, with similarity weights $\{w_1, \dots, w_K\}$. For each LLM m, we form a neighbor based estimate

$$\hat{s}_{\text{nb}}(q, m) = \frac{\sum_{i=1}^{K} w_i \, g(q_{di}, m)}{\sum_{i=1}^{K} w_i},$$

and interpolate it with a global prior expertise score prior(m), obtained by aggregating $g(q_d, m)$ over all training queries, to obtain the local score

$$s_{\text{local}}(q, m) = \alpha \, \hat{s}_{\text{nb}}(q, m) + (1 - \alpha) \, \text{prior}(m),$$

where $\alpha \in [0, 1]$ controls the trade-off between neighbor based evidence and the global prior.

In parallel, the *domain expertise* stage captures broader regularities in the query space. The discovery queries are clustered in the embedding space using k-means, and for each LLM m and cluster c we precompute a cluster profile value $\mu(m, K_c)$ summarizing how well m tends to perform on queries assigned to c. At inference time, the new query q is assigned to its closest cluster $K_c(q)$ in embedding space, and the domain-level score is defined as

$$s_{\text{domain}}(q, m) = \mu\big(m, Kc(q)\big),$$

which provides a coarse but robust estimate of the model's performance on that region of the query space. In this stage, a smoothing of the cluster estimates is done with the global expertise score to prevent the clusters to overfit. The final expertise score for a given query-LLM pair is a weighted combination of the local and domain level scores:

$$s_{\text{final}}(q, m) = \lambda \cdot s_{\text{local}}(q, m) + (1 - \lambda) \cdot s_{\text{domain}}(q, m),$$

where $\lambda \in [0, 1]$ controls the balance between query specific evidence and broader domain patterns. LLMs are then ranked in descending order of $s_{\text{final}}(q, m)$, producing the final submission ranking.

Implementation Details. Method 2 uses the Discovery Set, with its weak labels done with an LLM judge, and passes them directly to the encoder. Weak labeling is performed with the `olmo2-13b` model [8], which acts as an external judge and assigns a graded quality score in $\{0, 1, 2\}$ to each query-LLM pair. All queries are embedded using the `BAAI/bge-large-en` model [5], and embeddings are ℓ_2-normalized so that cosine similarity reduces to a dot product.

The query embeddings are clustered with k-means into $K_c = 40$ clusters to obtain the domain level profiles $\mu(m, c)$, similar to the clustering setup from Method 1. For the local component, we build a FAISS index [6] over the discovery query embeddings and, at inference time, retrieve the top-$K = 100$ nearest neighbors for each test query. The interpolation factor between the neighbor based estimate and the global prior is set to $\alpha = 0.9$, and the final combination weight between the local and domain level scores is set to $\lambda = 0.5$. These hyperparameters ($K = 100$, $K_c = 40$, $\alpha = 0.9$, $\lambda = 0.5$) are chosen based on Development Set performance and kept fixed for the reported runs.

5 Method 3: Clusters LLM Profiles

Method 3 is a straightforward cluster oriented approach that represents each LLM by a compact profile built from its past answers. The central assumption is that an LLM's expertise can be approximated by a small set of prototype answer vectors: if a new query is close to one of these prototypes in embedding space, the corresponding LLM is likely to provide a good answer. The method is fully automatic and internal: all profiles are derived from the Discovery Set only, and no manual adjustments are made after seeing the test queries.

In the first stage, all Discovery Set answers are embedded into the same semantic space as the queries using a sentence encoder. For each LLM m, its answer embeddings are clustered independently, producing a fixed number of centroids $\{\mathbf{c}_{m,1}, \dots, \mathbf{c}_{m,K}\}$ that summarize the typical responses of that model across the discovery queries. These centroids form a profile for the LLM: each centroid can be interpreted as a prototype capturing a recurring topic, style, or task that the model handles. Because only a small number of centroids is stored per model, the resulting profiles are compact and easy to inspect, while still capturing the main regions of answer-space where each LLM tends to operate.

At inference time, a new query q is encoded into the same embedding space to obtain $\mathbf{v}(q)$. For each LLM m, Method 3 computes the similarity between $\mathbf{v}(q)$ and all centroids in that model's profile, and retains the maximum similarity as the expertise score:

$$s_{\text{final}}(q, m) = \max_{1 \le j \le K} \text{sim}\big(\mathbf{v}(q), \mathbf{c}_{m,j}\big),$$

where $\text{sim}(\cdot, \cdot)$ denotes cosine similarity. Intuitively, this score measures how close the query is to the best-matching prototype for that LLM. Models whose centroid sets contain vectors near the query are considered more suitable; those whose prototypes lie far away are place down in the ranking. This simple max-over-centroids rule yields a single scalar score per LLM, and sorting these scores in descending order provides the final ranking. Although Method 3 does not explicitly use graded labels or weak supervision, it offers a strong baseline that is computationally efficient, interpretable, and aligned with the intuition that models can be summarized by a small set of characteristic answers.

Implementation Details. Method 3 uses only the Discovery Set answers to build LLM profiles, with no external labels or judges. All answers and queries are embedded using the `BAAI/bge-large-en` model [5], and embeddings are ℓ_2-normalized so that cosine similarity reduces to a dot product.

For each LLM, we apply k-means clustering independently to its normalized answer embeddings, using a fixed number of centroids per model, in our implementation, $K = 10$, which defines the prototype profile for that LLM. At inference time, each query is embedded once, and its score for a given LLM is computed as the maximum dot-product similarity between the query embedding and that LLM's centroids.

6 Method 4: Weakly Supervised MLP Classifier

Method 4 reframes LLM selection as a multi-class classification problem: given a query embedding, a neural model predicts a probability distribution over LLMs, indicating which model is most likely to provide a high-quality answer. Supervision comes from an external LLM judge [7], which labels the discovery data by indicating, for each query, which model produced the best answer. The goal is to learn a compact classifier that approximates this judge's routing behavior under the characteristics of the Discovery Set data.

The method begins with weak labeling. For every discovery query q and LLM m, the original query q and the corresponding answer $a(q, m)$ are shown to the external judge, which returns a graded quality score $\tilde{g}(q, m)$. Rather than using the full graded structure, Method 4 simplifies the problem by selecting, for each query, the LLM with the highest judged quality as the *"winner"*

$$m^*(q) = \arg\max_m \tilde{g}(q, m),$$

and encoding this as a single class label $y_q = m^*(q)$ for training. Each discovery query is then embedded using the shared sentence encoder to obtain a fixed-dimensional vector $\mathbf{v}(q)$, and the resulting (embedding, label) pairs form the training set.

On top of this representation, Method 4 trains a shallow multi-layer perceptron (MLP) classifier over LLM IDs. Let $\mathcal{M}$ denote the set of LLMs and $|\mathcal{M}|$ its size. The MLP defines a function

$$h : R^d \to R^{|\mathcal{M}|},$$

that maps a query embedding to a vector of logits $\mathbf{z}(q) = h(\mathbf{v}(q))$. These logits are converted into class probabilities via the softmax:

$$p_{\text{MLP}}(m \mid q) = \frac{\exp\big(z_m(q)\big)}{\sum_{m' \in \mathcal{M}} \exp\big(z_{m'}(q)\big)}.$$

Training uses standard cross-entropy loss over the weak labels, encouraging the network to assign high probability to the judge-selected model $m^*(q)$ for each query. Because the MLP is relatively small and operates on fixed query embeddings, it is efficient to train and fast at inference [9].

At inference time, the process is straightforward. A new query is embedded, passed through the trained MLP, and the resulting probabilities $p_{\text{MLP}}(m \mid q)$ are interpreted directly as expertise scores. The final score for a given query-LLM pair is

$$s_{\text{final}}(q, m) = p_{\text{MLP}}(m \mid q),$$

and LLMs are ranked in descending order of $s_{\text{final}}(q, m)$ to produce the final submission ranking. This approach trades some interpretability for efficiency, offering a compact parametric baseline for weakly supervised model selection.

Implementation Details. Method 4 uses the Discovery Set with weak labels obtained from an external LLM judge. Weak labeling is performed with the `olmo2-13b` model [8], which acts as an external judge and assigns a graded quality score in $\{0, 1, 2\}$ to each query-LLM pair. Queries are encoded using the `BAAI/bge-large-en` model [5], and embeddings are ℓ_2-normalized before being passed to the classifier.

The router is implemented as a shallow PyTorch MLP with input dimension 1024, two hidden layers of sizes 512 and 256 with ReLU activations. Adam optimizer with learning rate 10^{-3}, and 20 epochs for the MLP. All hyperparameters were tuned on the development set and then kept fixed for the submitted runs.

7 Method 5: Deep MLP Router

Method 5 extends the classification perspective of Method 4 by replacing the shallow MLP classifier with a higher capacity neural network that maps query embeddings directly to a probability distribution over LLMs. The underlying idea is the same: learn a function that, given a query, predicts which model is most likely to provide a high-quality answer, using weak supervision from an external LLM judge [7]. By increasing model capacity, via wider hidden layers and non-linear activations, plus dropout, Method 5 captures more subtle structure in the query space and improves expertise prediction [9].

As in Method 4, weak supervision is obtained by asking an external LLM judge to score the answers produced by each model for each discovery query. For every query q and LLM m, the judge returns a graded quality score $\tilde{g}(q, m)$. These scores per model are reduced to a single target label by selecting the LLM with the highest judged quality

$$m^*(q) \;=\; \arg\max_{m \in \mathcal{M}} \tilde{g}(q, m),$$

which is then encoded as a class index y_q for training. Queries are embedded using the shared sentence encoder to obtain vectors $\mathbf{v}(q) \in R^d$, and the resulting pairs $(\mathbf{v}(q), y_q)$ form the training set. The router is implemented as a feed-forward neural network

$$h : R^d \to R^{|\mathcal{M}|},$$

which maps the query embedding to a vector of logits $\mathbf{z}(q) = h(\mathbf{v}(q))$, one logit per LLM. These logits are converted into a probability distribution over LLMs using the softmax,

$$p(m \mid q) = \frac{\exp\bigl(z_m(q)\bigr)}{\sum_{m' \in \mathcal{M}} \exp\bigl(z_{m'}(q)\bigr)},$$

and the model is trained with cross-entropy loss to maximize the probability assigned to the judge-selected model $m^*(q)$ for each query:

$$\mathcal{L} = - \sum_{q \in \mathcal{Q}_{\text{disc}}} \log p\bigl(m^*(q) \mid q\bigr).$$

At inference time, Method 5 acts as a high capacity scoring function. A new query is embedded to obtain $\mathbf{v}(q)$, passed through the neural network method to produce logits, and the resulting softmax probabilities are interpreted directly as expertise scores. The final score for a given query–LLM pair is defined as

$$s_{\text{final}}(q, m) = p(m \mid q),$$

and LLMs are ranked in descending order of $s_{\text{final}}(q, m)$ to obtain the final run for that query.

Implementation Details. Method 5 uses the same Discovery Set and weak labels as Method 4, with scores $\tilde{g}(q, m)$ provided by an external LLM judge. Queries are embedded and normalized as in the previous methods.

The neural router is implemented as a customized PyTorch neuronal network with input dimension 1024, a hidden layer of size 512 with ReLU activation followed by dropout with rate 0.2, and an output layer of size $|\mathcal{M}|$ equal to the number of LLM classes. Training is performed using mini-batch optimization and the cross-entropy objective over the weak labels; a standard Adam optimizer with a fixed learning rate 10^{-3} is used. All hyperparameters are tuned on the development set and then kept fixed for the submitted runs.

8 Evaluation on the Development Set

We evaluated all five methods on the Development Set, which consists of 342 labeled queries with corresponding expertise judgments for each LLM. For every method, we produced a full ranking of LLMs per query and compared these rankings against the graded relevance labels using two standard metrics: nDCG@10 and MRR. The nDCG@10 metric rewards placing highly expert LLMs near the top of the ranking and accounts for graded relevance levels through a discounted cumulative gain formulation, while MRR focuses on the position of the first expert LLM, emphasizing how quickly a system can surface at least one strong model for a given query. Together, these metrics provide a complementary view: nDCG@10 reflects the overall quality of the top of the ranking and MRR is sensitive to early precision.

Method	nDCG@10	MRR
1	**0.5510**	0.7314
2	0.4519	**0.7362**
3	0.3675	0.6691
4	0.3968	0.6422
5	0.3861	0.6391

Table 1: Results for each method on the Development Set.

Table 1 summarizes the performance of the five strategies on the Development Set. Overall, Method 1 achieves the strongest nDCG@10, while Method 2 has the highest MRR, with a lower nDCG@10. The remaining methods, 3 to 5, stay behind in both nDCG@10 and MRR, suggesting that simple unsupervised profiles (Method 3) and purely parametric classifiers (Methods 4 and 5) struggle to match the effectiveness of hybrid and retrieval based designs in this setting.

Method 1 and Method 2 are the strongest in general, but they succeed for different reasons. Method 1 is fully unsupervised: it combines answer space cluster profiles with FAISS refinement per model, achieving the best nDCG@10 and a strong MRR. This suggests that it is particularly good at ranking the top few models in a fine-grained way, not only identifying a single good LLM but also ordering several strong candidates near the top. Method 2, by contrast, operates in the query space and leverages weak supervision via kNN and clustering. It's slightly higher MRR but much lower nDCG@10 indicates that it often places at least one relevant model at rank 1, but is less reliable at structuring the rest of the ranking, leading to shallower gains beyond the first position.

Method 3, the clustered LLM-profile approach, uses only centroid profiles over answer embeddings. Its performance is clearly below compared to Methods 1 and 2, but still shows that purely geometric structure in the answer space already encodes a useful notion of expertise. Methods 4 and 5 treat routing as direct multi-class prediction over LLM IDs using shallow and deeper neural classifiers, respectively. Both underperform the profile and retrieval-style methods, and the deeper MLP router does not improve over the shallow variant. In a fixed-data regime with a large number of classes and noisy weak labels, simply increasing model capacity does not translate into better ranking quality: the classifiers are likely affected by label sparsity, class imbalance, and the loss of graded information when collapsing scores to a single winner per query.

Overall, the results suggest that the most effective routers in this setting are those that exploit the geometric structure of the embedding space, either purely unsupervised (Method 1) or augmented with weak supervision from an external judge (Method 2). Purely unsupervised profiles (Method 3) provide an interpretable and efficient baseline but are limited by the absence of explicit quality signals, while purely parametric neural routers (Methods 4 and 5) struggle to exploit the available supervision effectively at this scale. The spectrum from Method 3 to Method 5 thus traces a trade-off between interpretability, reliance on labels, and modeling capacity, with the best-performing approaches remaining close to IR-style profiling and retrieval rather than collapsing the task into flat multi-class classification.

9 Conclusion

In summary, our experiments in the Million LLMs Track suggest that the methods benefit most from exploiting the geometric structure of the embedding space, with or without supervision. Among the five methods, the fully unsupervised hierarchical model in the answer space (Method 1) and the hybrid weakly supervised model in the query space (Method 2) achieve the strongest overall results on the development set. Method 1, which combines cluster profiles with FAISS-based answer retrieval, delivers the best nDCG@10 and a strong MRR, indicating that unsupervised profiles plus local answer level refinement are highly effective at ranking several strong LLMs near the top. Method 2, guided by weak labels from an external judge, has a slightly higher MRR but lower nDCG@10, suggesting that label driven kNN routing is good at finding at least one strong model early but less reliable at structuring the entire top of the ranking.

The remaining methods occupy different points in the trade-off space between supervision, interpretability, and modeling capacity. Method 3, which relies solely on unsupervised centroid profiles over answer embeddings, performs clearly below Methods 1 and 2 but still yields meaningful rankings, showing that purely geometric information already encodes a coarse notion of expertise. Methods 4 and 5, which cast routing as multi-class prediction over LLM IDs using shallow and deep neural classifiers, underperform the profile retrieval approaches.

Overall, these findings indicate that no single method is universally superior. The task of choosing the right LLM to answer a given query is a non-trivial one, and represents a balance between IR style profiling and parametric classification. Profile based methods offer clearer diagnostics and strong performance, especially the hierarchical design of Method 1. Our results point toward hybrid designs that more tightly integrate these paradigms, for example, by enriching neural routers with profile derived features or using uncertainty aware retrieval components informed by classifier outputs.

References

[1] Zhu, Y., Yuan, H., Wang, S., Liu, J., *et al. Large Language Models for Information Retrieval: A Survey.* arXiv preprint arXiv:2308.07107 (2023).

[2] Bommasani, R., Hudson, D. A., Adeli, E., *et al. On the Opportunities and Risks of Foundation Models.* arXiv preprint arXiv:2108.07258 (2021).

[3] Million LLMs Track. "TREC 2025: Million LLMs Track." https://trec-mllm.github.io/ (accessed 2025-11-30).

[4] Huang, J.-T., Sharma, A., Sun, S., Xia, L., *et al. Embedding-based Retrieval in Facebook Search.* arXiv preprint arXiv:2006.11632 (2020).

[5] BAAI, *BAAI/bge-large-en.* Hugging Face model. Available at: `https://huggingface.co/BAAI/bge-large-en` (accessed 2025-11-24).

[6] Douze, M., Guzhva, A., Deng, C., Johnson, J., *et al. The Faiss Library.* arXiv preprint arXiv:2401.08281 (2024).

[7] Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., *et al. A Survey on LLM-as-a-Judge.* arXiv preprint arXiv:2411.15594 (2024).

[8] Team OLMo. *OLMo 2 1124 13B.* allenai/OLMo-2-1124-13B, Hugging Face (2024). `https://huggingface.co/allenai/OLMo-2-1124-13B`

[9] Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., *et al. RouteLLM: Learning to Route LLMs with Preference Data.* arXiv preprint arXiv:2406.18665 (2024).