MODELLAKEFISHING: EFFICIENT RETRIEVAL OVER MILLION-SCALE MODEL LAKES

Anonymous authors
Paper under double-blind review

ABSTRACT

Open model lakes may contain millions of reusable models, making it costly to identify suitable models for a new dataset. This setting requires learning indexable representations of models from heterogeneous evidence, including model and dataset metadata, lineage, and historical evaluations. An important challenge is that this heterogeneous evidence is sparse and concentrated around popular models and benchmarks. We present ModelLakeFishing, a model-retrieval framework for queries that specify a target dataset, a prediction task, and an evaluation metric. It consolidates model lake metadata and historical evaluations into a model–dataset–task evidence graph, learns model and query embeddings with a relation-aware graph encoder, and indexes model embeddings with Hierarchical Navigable Small World (HNSW) search. At query time, HNSW retrieves 1k candidates without scoring every model in the lake. A task-level prior computed only from training-side evidence then reranks the candidates for the requested metric and returns the top 10 models. We evaluate the framework on a lake containing 3,016,439 models and 247,803 observed model–dataset performance pairs using three root-aware splits in which test performance edges are held out from representation learning and retrieval. ModelLakeFishing achieves a mean eligible-query `gold@10` of 0.2968, recovering the observed-best model among the ten returned models for 29.68% of eligible queries. This retains 93.47% of the `gold@10` achieved by an exhaustive baseline that applies the same scoring and reranking procedure to every model in the lake. Given precomputed query embeddings, candidate retrieval and reranking take 0.747 ms at the median and 1.102 ms at the 95th percentile. These results show that sparse relational evidence can be distilled into an indexable representation space for efficient retrieval over million-model lakes.[1]

1 INTRODUCTION

Recommendation over a large-scale model lake needs candidate discovery. Public model hubs now resemble *model lakes*: large collections of reusable models accompanied by model cards, configurations, lineage, and evaluation metadata (Pal et al., 2025; Dong et al., 2025). Hugging Face alone hosts more than three million candidate models across diverse tasks, architectures, and modalities. For a target dataset, trying every model is impractical, while computing a separate query-specific relevance score for every model still requires traversing the full lake. The immediate problem is therefore to find a small, useful shortlist before any candidate is downloaded or run.

Prior methods use different selection signals. Existing model-selection methods reduce the cost of assessing candidates in different ways. Feature-based transferability estimators such as LEEP (Nguyen et al., 2020), LogME (You et al., 2021), NLEEP (Li et al., 2021), and H-score (Bao et al., 2019) avoid full fine-tuning yet still execute each candidate on target-dataset samples. Other approaches use repository information to guide selection: TransferGraph uses learned graph representations in a performance regressor (Li et al., 2024); ModelLens combines metadata and evaluation records in a task- and metric-conditioned interaction scorer (Cai et al., 2026); and HuggingR[4]

[1]The data and code are available at https://anonymous.4open.science/r/ModelLakeFishing-8B73.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

retrieves candidates from model-card representations for natural-language requests before iterative refinement (Ma et al., 2025). Together, these works demonstrate interaction-based ranking and metadata-based retrieval, leaving open how sparse model–dataset evidence can support indexed candidate generation for dataset–task queries over the full lake.

Relation-aware learning makes evidence searchable. Historical performance records cover only a small fraction of the model lake, so direct evaluations alone cannot support candidate discovery throughout the collection. Model and dataset metadata provide descriptions that can be crawled and enriched (Li et al., 2023c); observed performance, dataset similarity, and model lineage provide complementary relations with different meanings. MODELLAKEFISHING brings these signals together in a heterogeneous model–dataset evidence graph and trains a relation-specific GraphSAGE encoder (Hamilton et al., 2017) to place models and dataset–task queries in a shared representation space. Unlike TransferGraph's downstream query–model performance regressor (Li et al., 2024), our retrieval score is a direct inner product between independently computed representations. This lets us index model representations offline with HNSW (Malkov & Yashunin, 2020), retrieve a bounded pool for a dataset–task query, and apply a historical same-task prior only when reranking that pool. Figure 1 summarizes this offline and query-time division.

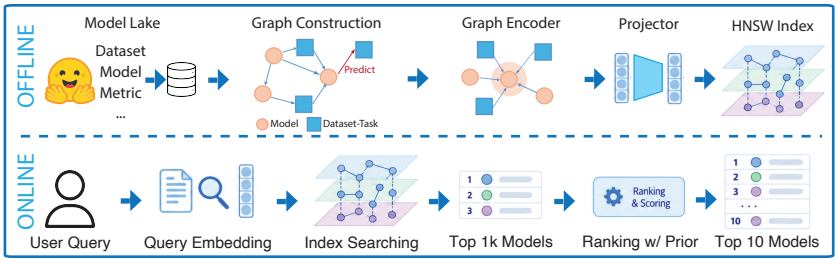

Figure 1: MODELLAKEFISHING overview. Everything above the dashed line is computed once, offline; everything below runs per query. The offline pipeline turns heterogeneous hub metadata and sparse performance evidence into a shared model–dataset–task embedding space and an HNSW index.

A shortlist retains strong candidates. We assess candidate discovery by withholding performance records for entire dataset roots (groups of related configurations of the same dataset)and asking whether the returned top ten include the best model *observed in those held-out records*. This observed-best recovery measure, `gold10`, is compared with a full-lake reference that uses the same learned representations and reranking score but considers every model. Indexed retrieval retains most of the reference's recovery while limiting retrieval and reranking to a bounded candidate pool. The lookup timing begins with a precomputed query representation, separating the cost of candidate discovery from query encoding.

Contributions. **(1) Candidate discovery through indexable representations.** We learn model and dataset–task representations from heterogeneous model–dataset evidence so that models across the full lake can be indexed once and retrieved without candidate-wise execution or query-conditioned scoring over every model. The retrieved candidate pool is then reranked using historical task evidence.

(2) A full-lake evidence graph and leakage-safe evaluation protocol. We construct and audit a multi-million-model evidence graph and split performance records by dataset root into training, validation, and test sets. Test performance records are used only for evaluation, not for representation learning or candidate discovery.

(3) Quality and efficiency at three-million-model scale. Across three dataset-root partitions, the indexed retrieval (candidate discovery) and reranking system achieves a mean `gold10` of 0.2968 and retains 93.47% of the `gold10` achieved by exhaustively scoring and reranking every model with the same representations and task prior. With query embeddings precomputed, candidate retrieval and reranking have a median latency of 0.747 ms and a 95th-percentile latency of 1.102 ms, with both values averaged across partitions. This compares with an average latency of 5.73 ms for the full-lake exhaustive scoring.

2 Related Work

Model lakes. Model management systems record experimental artifacts, versions, and intermediate results so that individual models can be reproduced, compared, or diagnosed (Vartak et al., 2018; Mei et al., 2022). Provenance research studies derivation and reuse across model versions (Mu et al., 2024). A model lake extends this scope from a controlled collection to reusable models, datasets, metadata, lineage, and evaluation evidence at repository scale (Pal et al., 2025; Garouani et al., 2024; Dong et al., 2025). For the massive model lakes we are considering, management and discovery are coupled since exhaustively considering every model for a tasks is infeasible. A useful model must be located before its artifacts can be inspected or evaluated, and evidence may be distributed across metadata, related repositories (for example code repositories), and historical evaluations. Public model lakes expose model cards and searchable repository metadata (Mitchell et al., 2019); Macaroni crawls and enriches metadata from public model zoos (Li et al., 2023c). Model-card documentation is nevertheless incomplete and uneven (Liang et al., 2024), limiting retrieval from lexical descriptions or normalized fields alone. An alternative to using documentation for search is to use behavior-based search (find models with a similar behavior) (Lu et al., 2023) or multimodal, meta-learning-based retrieval (Li et al., 2023b). These methods extend what a query can express, but the query itself still drives retrieval. Our setting is different: the evidence that makes a model retrievable is not only in the query but also in the lake's own relational structure — links among models, dataset–task pairs, observed performance, and model lineage, evidence that is sparse but structured. We ask how these relations can be distilled into an indexable search space over millions of candidates.

Model recommendation. Classical model selection ranks candidates through downstream validation or fine-tuning, but evaluating every model becomes prohibitive as the size of the model lake grows (Ding et al., 2018). Transferability estimators such as LEEP (Nguyen et al., 2020), LogME (You et al., 2021), NLEEP (Li et al., 2021), and H-score (Bao et al., 2019) avoid the expense of fully fine-tuning each candidate model on the target dataset, but still require running each candidate forward on target-dataset samples to extract its outputs or feature statistics. They therefore choose within an available pool, whereas our setting requires a shortlist from the full lake before models are downloaded or run. HuggingR^4 frames repository-scale selection as iterative reasoning, retrieval, refinement, and reflection (Ma et al., 2025). ModelSearch uses structural similarity between model-card tables to recommend models that are both relevant to the user's request and diverse (Dong & Miller, 2026). Historical model–dataset interactions provide a complementary basis for recommendation without candidate execution: TransferGraph learns from a graph of models, datasets, similarities, and observed performance (Li et al., 2024); ModelLens combines heterogeneous metadata with task- and metric-conditioned interaction evidence at larger scale (Cai et al., 2026). Both demonstrate that relational history improves recommendation, but their reported final relevance functions score individual query–candidate pairs rather than expose a query-independent model index. We ask how relational supervision can instead produce an ANN-searchable space over millions of models.

Large-scale indexing. Learned retrieval makes search indexable by encoding queries and candidates independently, then comparing their vectors with a fixed similarity function. Sentence-BERT makes candidate representations reusable for similarity search (Reimers & Gurevych, 2019); DPR encodes the corpus offline and uses approximate nearest-neighbor search to obtain a bounded candidate set for each online query (Karpukhin et al., 2020). This factorization is the interface between representation learning and indexing: unlike a joint query–candidate scorer, it lets candidate vectors be computed once rather than rescored across the full collection. Approximate nearest-neighbor systems reduce the cost of searching these spaces. HNSW provides high-recall search without an exhaustive scan in the typical query path (Malkov & Yashunin, 2020); Faiss provides complementary exact and approximate indexes, vector compression, and hardware-oriented implementations for large embedding collections (Douze et al., 2024). These methods assume relevance has already been expressed in a searchable vector space. Our problem lies at that preceding boundary: sparse model–dataset observations and heterogeneous graph relations do not directly define indexable model vectors. ModelLakeFishing uses inductive graph representation learning (Hamilton

et al., 2017) to produce query-independent model embeddings and compatible dataset–task embeddings, indexes models once with HNSW, and reserves query-conditioned task evidence for reranking only the retrieved set.

3 METHODOLOGY

3.1 PROBLEM FORMULATION AND OVERVIEW

Candidate Discovery Task. Let $\mathcal{D}$ denote the dataset–task query nodes in the evidence graph. Each query $d = (\text{normalize}(\text{dataset}), t) \in \mathcal{D}$ identifies a canonicalized dataset and a prediction task t Different tasks on the same dataset can therefore induce different model rankings. The objective is to return k models from the candidate model lake $\mathcal{M}$ that recover strong candidates. We judge strong candidates to be candidates that are ranked high in held-out historical evaluations. Table 1 summarizes the notation used.

Table 1: Notations.

Notation	Description	Notation	Description
$\mathcal{M}$	Model lake.	$z_d,\ z_m$	Query and model embeddings.
$N = \lvert\mathcal{M}\rvert$	Model lake size.	$s_\theta(d, m)$	Retrieval score.
$\mathcal{D}$	Dataset–task nodes.	$\widehat{\mathcal{P}}_K(d)$	Top-K retrieved models.
d	Query node.	$p_t(m)$	Same-task prior.
y_{dm}	Observed performance for (d, m).	$r(d, m)$	Final reranking score.

Evidence Graph Construction. Historical evaluation records report how particular models performed on particular dataset–task pairs, but cover only a small part of the model lake. We organize these records into a graph with model nodes and dataset–task nodes. Each recorded evaluation connects a model to the corresponding dataset–task node; dataset simialrity and model lineage provide the other relations. This graph supplies the context from which we learn model and query representations and is described in Section 3.2

Learning and lookup are separate. MODELLAKEFISHING constructs an evidence graph and learns unit-normalized model and query representations offline. At query time, it retrieves a bounded candidate pool from an index of model representations and reranks that pool using a same-task historical prior. We collect historical model evaluation results from model cards and existing evaluation collections. Each record identifies a model m, a dataset–task pair d, and a measured performance score of m on d. After preprocessing, the score becomes y_{dm}, and the corresponding pair defines a performance edge. We partition these records by dataset into training, validation, and test sets. The nodes, metadata, dataset-similarity edges, and model-lineage edges remain the same. During training, 70% of the training performance edges are included in the message-passing graph. The remaining 30% are held out of the graph, and their scores provide supervision for the ranking loss. Validation and test performance edges are excluded from the training graph, and their scores are not used for training. During testing, we measure recommendation quality on the test set. Using the trained parameters, we compute model and query representations with all training and validation performance edges as graph context, then retrieve and rerank models for each test query. Test performance edges are held out of this graph in both directions; their recorded scores provide the ground truth for evaluating the returned rankings.

3.2 EVIDENCE GRAPH CONSTRUCTION

Node: Sparse records supply supervision. The evidence corpus combines Hugging Face crawls dated August 18, 2026 with five historical evaluation graphs. We canonicalize model IDs and attach dataset cards only through exact or unambiguous repository matches; otherwise, the dataset name and task recorded in the evaluation record provide the description. We create a node for every model in the crawl or referenced in the crawl. The referenced models include 12,680 absent from the crawl but unambiguously referenced crawled model or dataset cards. We assign each node a fixed integer ID and use it consistently for its feature row, graph-edge endpoints, and learned embedding; model IDs also label entries in the search index.

We create a dataset–task node only when that pair appears in the evaluation records, whether or not a dataset card is available. We group performance observations by dataset, task, and metric and select one primary metric for each dataset–task node. Repeated observations are then aggregated, normalized, and oriented so that higher values indicate better performance. The merged corpus contains 247,803 observed model–dataset performance pairs with values notated as y_{dm}, many fewer than the number of candidate models. Appendix B.2 specifies metric selection, source resolution, and per-node sampling.

Edges: Typed relations connect evidence. The graph has model and dataset–task nodes. Its five directed relations comprise observed performance and its reverse, dataset similarity, and parent-to-derivative lineage and its reverse. Performance edges represent recorded evaluations, while similarity connects each dataset to its ten nearest neighbors by descriptor cosine and lineage connects parent and derivative model nodes using resolved parent-model references in the model metadata. These relations connect sparse observations with metadata and other structural evidence without treating lineage or similarity as measured performance.

3.3 Relation-Aware Representation Learning

Metadata anchors both node types. Model and dataset–task inputs combine a name hash and a text embedding from a sentence transformer (Wang et al., 2020) with learned type-specific lookup features. Model text describes the repository name, family, and parameter size when known; dataset text uses a matched card when available and otherwise uses its name and task. Dataset nodes also include root-level statistics. The resulting model and dataset–task inputs have 480 and 486 dimensions, respectively. Node inputs contain neither performance statistics nor gold-eligibility indicators; feature construction is detailed in Appendix B.2.

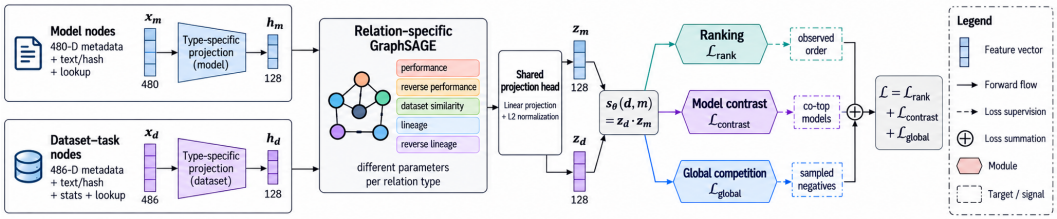

Figure 2: Representation learning and training. Model and dataset–task features pass through type-specific projections, a relation-specific GraphSAGE encoder, and a shared projection head. Ranking and global losses compare query–model pairs; model contrast compares model pairs.

Relation-aware aggregation yields indexable vectors. Indexable retrieval requires model vectors that do not depend on the incoming query. We use a relation-specific GraphSAGE layer (Hamilton et al., 2017) to independently compute model and query representations. Type-specific projections map both node types into a shared hidden space, followed by relation-aware neighborhood aggregation and a shared output head. The resulting unit-length representations are scored by inner product, $s_\theta(d, m) = z_d^\top z_m$. The detailed encoder formulation is provided in Appendix B.5. The key property of this architecture is that model and query representations are computed independently. This means model vectors can be indexed once offline and searched for any query without rescoring the full lake. The design achieves this by sharing a single output head across both node types, placing models and queries in the same vector space, while keeping aggregation unweighted. Performance values provide supervision and dataset similarity determines graph topology, but neither scales the messages passed between neighbours. Hash and text features remain frozen; lookup tables, projections, gates, and the shared head are learned.

3.4 Training with Held-Out Evidence

Root splits control evidence visibility. A dataset root groups configurations of the same dataset. All nodes sharing a dataset root belong to the same training, validation, or test partition, so related configurations do not cross evaluation boundaries. A disjoint portion of training performance edges supplies supervision; the remaining training edges carry messages. Let E_{tr} denote performance edges from training roots. Validation message passing can use all training edges, and test message passing can also use validation edges, but neither graph contains the performance edges of the roots

it evaluates. Reverse performance edges follow the same rule, while dataset similarity and lineage remain fixed. Reported indexes and query vectors use the test message graph, making this transductive retrieval with held-out performance evidence. The exact edge sets and split proportions appear in Appendix B.3.

Training aligns the retrieval score. We jointly optimize local ranking, model contrast, and whole-lake competition with equal weight.

$$\mathcal{L} = \mathcal{L}_{\text{rank}} + \mathcal{L}_{\text{contrast}} + \mathcal{L}_{\text{global}} \tag{1}$$

All three objectives use the unit-normalized representations in Eq. 8. Their temperatures τ_{rank}, τ_{contrast}, and τ_{global} are listed in Appendix B.3.

Local ranking preserves observed order. For each supervised query d, we retain preference pairs $\mathcal{P}_d \subseteq \{(i,j) : y_{di} - y_{dj} > 0.02\}$. RankNet (Burges et al., 2005) penalizes retrieval scores that reverse their observed performance order.

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{D}_B|} \sum_{d \in \mathcal{D}_B} \frac{1}{|\mathcal{P}_d|} \sum_{(i,j) \in \mathcal{P}_d} \text{softplus}\left(-\frac{s_\theta(d,i) - s_\theta(d,j)}{\tau_{\text{rank}}} \right) \tag{2}$$

Here $\mathcal{D}_B$ contains batch queries with retained pairs. We keep at most 256 pairs per query; when capped, half are the most inverted under the current model and half are sampled from the remainder. Averaging within queries limits the influence of densely observed datasets.

Model contrast links strong models. For a query d with n_d training-visible observations, let H_d contain its top $k_d = \max(1, \text{round}(0.1 n_d))$ models. Distinct models sharing membership in any H_d form positive pairs. Let $\mathcal{P}_i$ be the batch positives for anchor i, and let $\mathcal{A}$ contain anchors with nonempty $\mathcal{P}_i$. For each anchor, we draw 256 batch models uniformly with replacement and remove self IDs and positives, leaving a negative multiset $\mathcal{N}_i$. With $\phi(i,j) = z_i^\top z_j / \tau_{\text{contrast}}$, the contrastive objective is

$$\mathcal{L}_{\text{contrast}} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \left[\log \left(\sum_{p \in \mathcal{P}_i} e^{\phi(i,p)} + \sum_{n \in \mathcal{N}_i} w_{in} e^{\phi(i,n)} \right) - \frac{1}{|\mathcal{P}_i|} \sum_{p \in \mathcal{P}_i} \phi(i,p) \right] \tag{3}$$

The negative weight w_{in} is two for models in the anchor's lineage component and one otherwise: shared ancestry alone does not imply similar performance.

Global competition covers the lake. Local batches expose each query to only a small portion of the model lake. We draw additional negatives from $q(m) = \frac{1}{2} q_{\text{deg}}(m) + \frac{1}{2} q_{\text{lab}}(m)$, where $q_{\text{deg}}(m) \propto (\deg(m) + 1)^{0.75}$ over all models and q_{lab} is uniform over models with positive training degree. Degree counts observations in E_{tr} only. Each step samples 128 positive-bearing queries and 256 negatives per query with replacement; removing members of H_d leaves the multiset S_d. Let $a_{dm} = s_\theta(d,m)/\tau_{\text{global}}$ and let $\mathcal{Q}_B$ denote the sampled queries. The sampled-softmax objective is

$$\mathcal{L}_{\text{global}} = \frac{1}{|\mathcal{Q}_B|} \sum_{d \in \mathcal{Q}_B} \left[\log \left(\sum_{p \in H_d} e^{a_{dp}} + \sum_{n \in S_d} e^{a_{dn} - \log q(n)} \right) - \frac{1}{|H_d|} \sum_{p \in H_d} a_{dp} \right] \tag{4}$$

The $-\log q(n)$ correction applies only to sampled negatives. This objective uses a fresh full-message-graph forward pass with edge dropout. Mini-batches are rooted at disjoint supervision edges, whose labels are never dropped. Optimization settings and split seeds are in Appendix B.3.

3.5 Indexed Retrieval and Reranking

Indexed Candidate Retrieval. The score in Eq. 8 uses model vectors that are independent of the query, so we add every model vector to an inner-product HNSW index (Malkov & Yashunin, 2020) offline. Given query vector z_d, the index retrieves $K = 10^3$ nearby models from the full lake, forming the candidate pool $\widehat{\mathcal{P}}_K(d)$. Only models in this pool proceed to task-prior reranking. HNSW is an approximate search algorithm whose thoroughness is controlled by a parameter ef_search where larger values explore more of the index and find neighbours closer to the true nearest neighbours, at the cost of additional search time. We set ef by comparing HNSW's retrieved candidates

against exact brute-force nearest neighbours — finding the smallest ef_search that recovers nearly all exact neighbours — rather than tuning it directly against the gold evaluation labels. Details are in Appendix C.3.

Historical Evidence-Based Reranker. For task t and model m, let n_{tm} and A_{tm} be the count and sum of normalized evidence values on training and validation roots. Canonical task strings define these groups; the scored test root and its sibling configurations are excluded. We use a fixed shrinkage prior and fuse it with the normalized retrieval score. Unseen task–model pairs receive no historical bonus, and no query-wise min–max rescaling is applied. Only retrieved candidates receive prior lookups; a fixed label-free model order breaks exact ties before returning the top ten. This reranker is deterministic and cannot recover a model omitted from the candidate pool. The exact prior and score-fusion formulation is given in Appendix B.4.

4 EXPERIMENTS

For each test dataset and task, we recommend ten models. We check whether the highest-scoring model in the available test records is among those ten. We also measure how long it takes to search the model lake. We repeat the test using three different train/validation/test splits. The two tables below report results for one of these splits. Appendix C gives more details.

4.1 EVALUATION PROTOCOL

Dataset: Evaluation queries and data splits. The lake is sparse: at least 92% of the 3,016,439 candidate models have no recorded performance pair, and model size is unknown for 72% of candidates. This sparsity is the central challenge that ModelLakeFishing is designed to address. Each query names a dataset and a task already in the graph. A dataset root groups related configurations of one dataset. We divide the data into training, validation, and test groups in three different ways and keep every configuration of a dataset in the same group. We retrain the model for each split using the same initialization seed. For a test query, the model can see the dataset, task, and their metadata, but it cannot see the links that record how models performed on that dataset. We remove those links in both graph directions and use their scores only to check the returned list. The task prior also uses no results from the test dataset or its related versions. The corpus is constructed as follows. The snapshot has 1,008,417 dataset records. We make graph nodes only for dataset–task combinations with recorded evaluations. After normalizing dataset names, grouping records by dataset and task, and choosing one metric for each group, we have 18,729 dataset–task nodes. Removing duplicate or conflicting records and limiting records per node leaves 247,803 model–dataset performance pairs. Dataset cards match 3,928 nodes (21%). For testing, we exclude reinforcement-learning tasks and placeholder dataset names, because reinforcement-learning reward metrics have no consistent orientation, and placeholder names indicate unreliable dataset provenance. Each test query must have at least three models with recorded scores that are not all equal. We must also know whether higher or lower is better for the metric. The three different splits contain 1,476, 1,101, and 1,545 test queries from 730, 495, and 473 dataset roots, respectively. The splits differ in size because the 70/10/20 edge-proportion target is applied to dataset roots of varying size, producing unequal test query counts. Appendix B.1 lists the six input sources. Appendices B.2 and B.3 give the construction and training settings.

Baselines. We compare three search methods using the same learned vectors and task prior. The task prior uses a model's past results on other datasets with the same task. The first method (EXHAUSTIVE) scores and ranks all 3,016,439 models. The second checks similarity to every model (EXACT), keeps the closest 1k, and reranks that smaller set. The third (MODELLAKEFISHING) uses the HNSW index to find 1k nearby models and reranks them. Comparing the first two shows the effect of limiting the candidate set; comparing the last two shows the extra effect of approximate nearest-neighbour search. LEEP, LogME, NLEEP, and H-score compute a score from the target data for each candidate. TransferGraph and ModelLens also score each query–candidate pair. However, these approaches do not scale to the massive data lake we consider here.

Lookup time. We start the timer after the query vector has been computed. We stop it after HNSW search, task-prior lookup, combining the scores, and sorting. The time does not include computing

the query vector, loading the index, or sending data over a network. The reported full-lake scan was timed over a different set of steps, so its time cannot be used to calculate an online speedup. Appendix C.4 gives the other recorded times and memory measurements.

4.2 Experimental Results

For each test query d, let g_d be the highest-scoring model among those with recorded test results. We first adjust each metric so that higher scores are always better. A query is a *hit at* k when g_d appears among the first k returned models. Gold@k is the share of test queries that are hits.

$$\big(\texttt{gold@}k\big)_s = \frac{1}{|\mathcal{Q}_s|} \sum_{d \in \mathcal{Q}_s} \mathbf{1}[g_d \in R_k(d)] \tag{5}$$

We mainly report $k = 10$ and average the three split results with equal weight. This measure checks the best model we have tested, not whether an untested model might be better. Appendix C.2 defines the other measures in Figure 3(d). The results reported in the two tables below are based on a split with 1,476 queries. MRR@10 averages 1 divided by the rank of the best tested model when it appears in the top ten; it reports zero otherwise. Gold coverage (Gold cov.) at 1k is the share of queries for which the best tested model enters the 1k candidate pool before reranking.

Run[2]	Gold@1	Gold@5	Gold@10	MRR@10	Gold cov.@1k
Ablation results : w/ task prior					
ModelLakeFishing (full)	**15.8537**	29.6070	33.1978	21.7172	53.9295
w/o graph encoder	13.4146$_{\downarrow 2.44}$	27.3713$_{\downarrow 2.24}$	29.1328$_{\downarrow 4.07}$	19.3052$_{\downarrow 2.41}$	39.4986$_{\downarrow 14.43}$
w/o ranking	15.5827$_{\downarrow 0.27}$	**30.0136**$_{\uparrow 0.41}$	**35.0271**$_{\uparrow 1.83}$	**21.8393**$_{\uparrow 0.12}$	**58.6721**$_{\uparrow 4.74}$
w/o model contrast	14.7019$_{\downarrow 1.15}$	26.6260$_{\downarrow 2.98}$	29.8103$_{\downarrow 3.39}$	19.8643$_{\downarrow 1.85}$	47.9675$_{\downarrow 5.96}$
w/o global competition	0.0000$_{\downarrow 15.85}$	0.0000$_{\downarrow 29.61}$	0.0000$_{\downarrow 33.20}$	0.0000$_{\downarrow 21.72}$	0.0000$_{\downarrow 53.93}$
Ablation results : wo/ task prior					
ModelLakeFishing (full)	**1.8970**	6.7751	12.0596	**4.3722**	53.9295
w/o graph encoder	1.3550$_{\downarrow 0.54}$	**7.7236**$_{\uparrow 0.95}$	11.6531$_{\downarrow 0.41}$	4.0667$_{\downarrow 0.31}$	39.4986$_{\downarrow 14.43}$
w/o ranking	1.3550$_{\downarrow 0.54}$	7.1816$_{\uparrow 0.41}$	**13.2114**$_{\uparrow 1.15}$	4.0573$_{\downarrow 0.31}$	**58.6721**$_{\uparrow 4.74}$
w/o model contrast	1.2195$_{\downarrow 0.68}$	3.2520$_{\downarrow 3.52}$	5.5556$_{\downarrow 6.50}$	2.1613$_{\downarrow 2.21}$	47.9675$_{\downarrow 5.96}$
w/o global competition	0.0000$_{\downarrow 1.90}$	0.0000$_{\downarrow 6.78}$	0.0000$_{\downarrow 12.06}$	0.0000$_{\downarrow 4.37}$	0.0000$_{\downarrow 53.93}$
Baseline comparison					
Full model + task prior	**15.8537**	**29.6070**	**33.2656**	**21.7284**	**53.8618**
Full model w/o task prior	1.8970$_{\downarrow 13.96}$	6.7751$_{\downarrow 22.83}$	12.0596$_{\downarrow 21.21}$	4.3722$_{\downarrow 17.36}$	**53.8618**$_{=}$
Frozen MiniLM	0.9485$_{\downarrow 14.91}$	2.4390$_{\downarrow 27.17}$	3.9973$_{\downarrow 29.27}$	1.6166$_{\downarrow 20.11}$	15.1084$_{\downarrow 38.75}$
Popularity	0.0000$_{\downarrow 15.85}$	0.0000$_{\downarrow 29.61}$	0.0000$_{\downarrow 33.27}$	0.0000$_{\downarrow 21.73}$	0.9485$_{\downarrow 52.91}$
FM (metadata-adapted)	0.0000$_{\downarrow 15.85}$	0.0000$_{\downarrow 29.61}$	0.0000$_{\downarrow 33.27}$	0.0000$_{\downarrow 21.73}$	0.1355$_{\downarrow 53.73}$

Main result On the seed-0 split, the full HNSW system achieves 33.20% gold@10 and 53.93% gold coverage@1k (first table). Removing the task prior leaves pool coverage unchanged but lowers gold@10 to 12.06%, showing that the prior improves ranking within the retrieved pool. Removing the graph encoder or model contrast lowers gold@10 to 29.13% and 29.81%, respectively; removing global competition yields zero coverage on this split. The ranking-loss ablation moves in the opposite direction, raising gold@10 to 35.03%, so this split does not establish a benefit from that loss. We compare three baselines without a task prior: frozen MiniLM (Wang et al., 2020), a pretrained text encoder that ranks models by query–model cosine similarity; popularity, which ranks by downloads; and a metadata-adapted Factorization Machine (FM) (Li et al., 2023a), which predicts performance using pairwise metadata feature interactions. In the exact top-1k comparison, the full model without the prior achieves 12.06% gold@10, versus 4.00% for MiniLM and 0% for both popularity and FM. Adding the prior raises gold@10 to 33.27%.

[2]Values are percentages. Subscripts show changes relative to the first row. The three methods use no task prior. FM is a metadata adaptation fitted on 198,216 training and validation observations; the full encoder is fitted on the training partition.

Comparison to Baselines. Figure 3 extends the retrieval comparison to three dataset-root splits and reports other evaluation views. Panel (a) shows that (averaging over all splits), the mean gold@10 (best observed hit rate) changes from 31.74% with EXHAUSTIVE to 29.70% with EXACT and 29.68% with MODELLAKEFISHING. This shows that almost all of the loss comes from limiting the pool, rather than from approximate index search. Panel (b) reports the hit-rate retention for all three methods. Panel (c) applies alternative success measures to the same returned lists (reported only for MODELLAKEFISHING). Panels (d–e) report lookup time after query encoding, with a median of 0.747 ms and a 95th percentile of 1.102 ms. Panels (f–h) report training, other processing intervals, and memory separately (time not included in the query time).

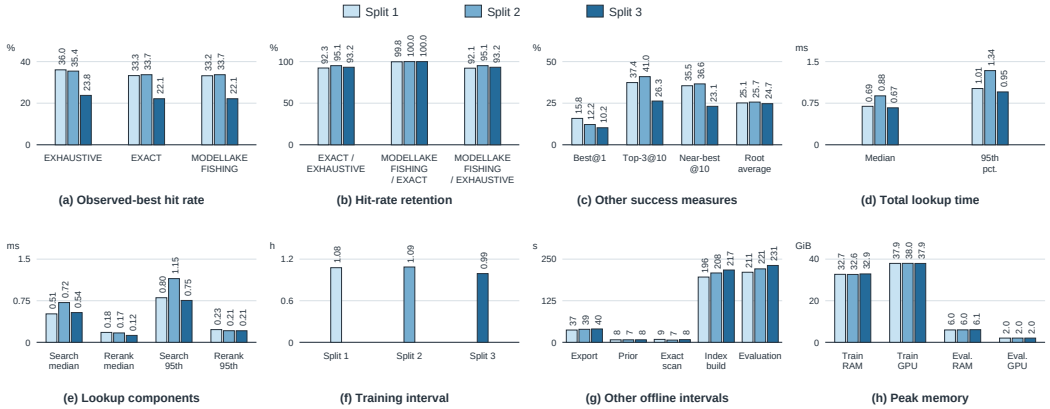

Figure 3: Results for three dataset-root splits. Within each group, bars follow the legend from left to right; numbers above bars are rounded.

5 CONCLUSION

Model lakes containing millions of reusable models present a retrieval challenge that exhaustive scoring cannot meet. Identifying suitable models for a new dataset requires learning from evidence that is sparse, heterogeneous, and unevenly distributed across a corpus too large to evaluate candidate by candidate. MODELLAKEFISHING addresses this by learning indexable representations from a model–dataset evidence graph. We combine performance records, model cards, dataset descriptions, and model lineage into a structured graph. We index embeddings with HNSW so that retrieval requires no candidate-wise execution at query time.

Evaluated on a lake of 3,016,439 models with a leakage-safe, root-aware protocol across three dataset partitions, MODELLAKEFISHING achieves a mean gold@10 of 39% — recovering the observed best model in the top ten for nearly 30% of eligible queries. It retains 93% of the gold@10 achieved by exhaustive scoring, with a median query latency of 0.747ms after query encoding. Nearly all of the quality gap relative to exhaustive search comes from limiting the candidate pool to 1,000 models; the HNSW approximation itself costs less than 0.1 percentage points of gold@10. The task-level prior is the dominant reranking signal: removing it drops gold@10 from 33% to 12% on the seed-0 split, while pool coverage remains unchanged, confirming that the prior improves ranking within the retrieved set rather than retrieval itself.

These results demonstrate that an ANN-searchable representation space over millions of models can be learned from relational supervision alone, without executing any candidate model on the target dataset. The evaluation measures recovery of historically observed models under sparse supervision; the true best model for a query may be unobserved and unreachable by any retrieval system operating over recorded evidence alone. Competitors do not scale to the size of graph we have considered, but a full comparison with methods that score individual query–candidate pairs, such as TransferGraph and ModelLens, on a matched candidate subset remains an important open experiment

AI USE STATEMENT

Generative AI tools were used to assist with language editing, code development and debugging, figure and LaTeX formatting, and brainstorming during the preparation of this work. The research methodology, experimental design, evaluation, and conclusions were developed and verified by the authors. All AI-assisted content and code were reviewed and validated by the authors, who take full responsibility for the final contents of the paper.

REFERENCES

Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. An information-theoretic approach to transferability in task transfer learning. In *IEEE International Conference on Image Processing (ICIP)*, pp. 2309–2313, 2019.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 89–96, 2005. doi: 10.1145/1102351.1102363.

Rui Cai, Weijie Jacky Mo, Xiaofei Wen, Qiyao Ma, Wenhui Zhu, Xiwen Chen, Muhao Chen, and Zhe Zhao. ModelLens: Finding the best for your task from myriads of models. *arXiv preprint arXiv:2605.07075*, 2026.

Jie Ding, Vahid Tarokh, and Yuhong Yang. Model selection techniques: An overview. *IEEE Signal Processing Magazine*, 35(6):16–34, 2018. doi: 10.1109/MSP.2018.2867638.

Zhengyuan Dong and Renée J Miller. Diversed model discovery via structured table discovery. *arXiv preprint arXiv:2605.22766*, 2026.

Zhengyuan Dong, Victor Zhong, and Renée J. Miller. Modeltables: A corpus of tables about models. *CoRR*, abs/2512.16106, 2025. doi: 10.48550/ARXIV.2512.16106. URL https://doi.org/10.48550/arXiv.2512.16106.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024.

Moncef Garouani, Franck Ravat, and Nathalie Vallès-Parlangeau. Model lake : A new alternative for machine learning models management and governance. In Mahmoud Barhamgi, Hua Wang, and Xin Wang (eds.), *Web Information Systems Engineering - WISE 2024 - 25th International Conference, Doha, Qatar, December 2-5, 2024, Proceedings, Part IV*, volume 15439 of *Lecture Notes in Computer Science*, pp. 133–144. Springer, 2024. doi: 10.1007/978-981-96-0573-6_10. URL https://doi.org/10.1007/978-981-96-0573-6_10.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1024–1034, 2017.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550.

Hao Li, Charless Fowlkes, Hao Yang, Onkar Dabeer, Zhuowen Tu, and Stefano Soatto. Guided recommendation for model fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3633–3642, 2023a. doi: 10.1109/CVPR52729.2023.00354.

Yandong Li, Xuhui Jia, Ruoxin Sang, Yukun Zhu, Bradley Green, Liqiang Wang, and Boqing Gong. Ranking neural checkpoints. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2663–2673, 2021.

Zhaotian Li, Binhang Qi, Hailong Sun, and Xiang Gao. Automrm: A model retrieval method based on multimodal query and meta-learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1228–1237, 2023b.

Ziyu Li, Henk Kant, Rihan Hai, Asterios Katsifodimos, and Alessandro Bozzon. Macaroni: Crawling and enriching metadata from public model zoos. In *International Conference on Web Engineering*, pp. 376–380. Springer, 2023c.

Ziyu Li, Hilco Van Der Wilk, Danning Zhan, Megha Khosla, Alessandro Bozzon, and Rihan Hai. Model selection with model zoo via graph learning. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1296–1309. IEEE, 2024. doi: 10.1109/ICDE60146.2024.00088.

Weixin Liang, Nazneen Rajani, Xinyu Yang, Ezinwanne Ozoani, Eric Wu, Yiqun Chen, Daniel Scott Smith, and James Zou. What's documented in ai? systematic analysis of 32k ai model cards, 2024.

Daohan Lu, Sheng-Yu Wang, Nupur Kumari, Rohan Agarwal, Mia Tang, David Bau, and Jun-Yan Zhu. Content-based search for deep generative models. In *SIGGRAPH Asia 2023 Conference Papers*, pp. 1–12, 2023.

Shaoyin Ma, Chenggong Hu, Huiqiong Wang, Li Sun, Mingli Song, and Jie Song. HuggingR^4: A progressive reasoning framework for discovering optimal model companions. *arXiv preprint arXiv:2511.18715*, 2025.

Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4): 824–836, 2020.

Songzhu Mei, Cong Liu, Qinglin Wang, and Huayou Su. Model provenance management in mlops pipeline. In *Proceedings of the 2022 8th International Conference on Computing and Data Engineering*, pp. 45–50, 2022.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, pp. 220–229, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361255. doi: 10.1145/3287560.3287596. URL https://doi.org/10.1145/3287560.3287596.

Xin Mu, Yu Wang, Yehong Zhang, Jiaqi Zhang, Hui Wang, Yang Xiang, and Yue Yu. Model provenance via model dna. In *ECAI 2024*, pp. 1043–1050. IOS Press, 2024.

Cuong V. Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. LEEP: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning (ICML)*, pp. 7294–7305, 2020.

Koyena Pal, David Bau, and Renée J. Miller. Model lakes. In *EDBT*, pp. 985–995. OpenProceedings.org, 2025.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Manasi Vartak, Joana M. F. da Trindade, Samuel Madden, and Matei Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pp. 1285–1300, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450347037. doi: 10.1145/3183713.3196934. URL https://doi.org/10.1145/3183713.3196934.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2002.10957.

Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. LogME: Practical assessment of pre-trained models for transfer learning. In *International Conference on Machine Learning (ICML)*, pp. 12133–12143, 2021.

A Model Lake Visualization



Figure 4: The model lake. A UMAP projection of the 3,016,439 learned model representations in the shared 128-dimensional retrieval space, colored by model family; a small peripheral portion lies outside the plotted frame. The overlaid held-out `broad tweet corpus` / *token-classification* example marks the query (plus sign at the centroid of its dense top-64 models), its gold model (star), and the ten returned models (numbered circles), showing how one retrieval is situated within the full lake.

B Supplementary Method Details

B.1 Evidence Sources

Table 2 records the contribution of each input evaluation collection. These are data inputs, not baseline methods or independent runs of the proposed retriever. The names are provenance IDs: for example, `diverse_zoo` denotes one historical evaluation collection, not a method called "Diverse Zoo." Its underlying collection is not further characterized in the available evidence.

Table 2: Retained model–dataset performance pairs by input source ID after deduplication, conflict resolution, and per-node capping.

Input source ID	Pairs
`modellens_v2`	117,898
`hf_model_index`	73,670
`d0_v1_5`	45,992
`hf_effective`	5,223
`a_ctrl_2000m`	3,670
`diverse_zoo`	1,350
Total	247,803

B.2 Evidence Corpus and Node Features

Construction of the evaluated lake. The frozen hub snapshot contains 3,003,759 model records and 1,008,417 dataset records. Historical evaluations contribute 12,680 additional model IDs to the candidate universe. The parser reads 2,158,375 native metric rows, retains 2,097,081 finite and parseable rows, and reduces these to 1,435,162 after within-key median deduplication. Across the six retained evidence sources, the merge starts from 531,958 rows, removes 5,102 within-source duplicates, resolves 1,502 cross-source conflicts, and applies per-dataset capping. The source breakdown is given in Table 2; it describes data provenance rather than competing retrieval methods.

Metric and feature processing. Native evaluation records are grouped by dataset, task, and full metric name, retaining cutoff distinctions. Invalid numerical values are discarded; repeated model observations are median-aggregated and min–max normalized within each group, with constant groups assigned 0.5. Lower-is-better values are oriented as $y = 1 - \bar{v}$. Each node selects one primary metric, preferring known direction, then reward, then unknown; observation count and lexical order break ties. Reward/unknown values (2.03% of retained performance edges) remain numerical evidence for learning and priors but cannot define evaluation gold. Historical sources retain their stored orientation: duplicate (source, node, model) records are median-aggregated and normalized within each (source, node) group. Fixed source priority resolves overlapping model–node pairs, and deterministic weight-stratified sampling caps each node at 200 observations.

Node inputs combine a frozen 64-dimensional name hash with a 384-dimensional embedding from `all-MiniLM-L6-v2`. Model text contains the cleaned repository name, canonical family, and a parameter-size phrase when known. Dataset text contains its cleaned name and task, matched card task categories, up to ten non-colon tags, and at most 400 description characters when a card match is reliable. Dataset nodes append a ten-slot statistics block whose only nonzero coordinate is the seventh, $\log(1 + r_d)$, where r_d counts frozen dataset–task nodes sharing the same root. Learned size and family lookups add 16 dimensions each to model inputs, with explicit unknown categories; task-type, class-count, and arity lookups add 16, 8, and 4 dimensions to dataset inputs. The resulting input dimensions are 480 and 486.

The model feature matrix has 448 columns. Dataset–task features have 458 columns; seven performance-derived input columns are zeroed in the evaluated configuration. The graph relation counts appear in Figure 5. For training, each dataset keeps its ten nearest dataset neighbors from a stored top-20 graph, and those links contribute equally; the stored count reflects the larger construction graph. These are properties of the fixed input and training recipe, not component-removal experiments.

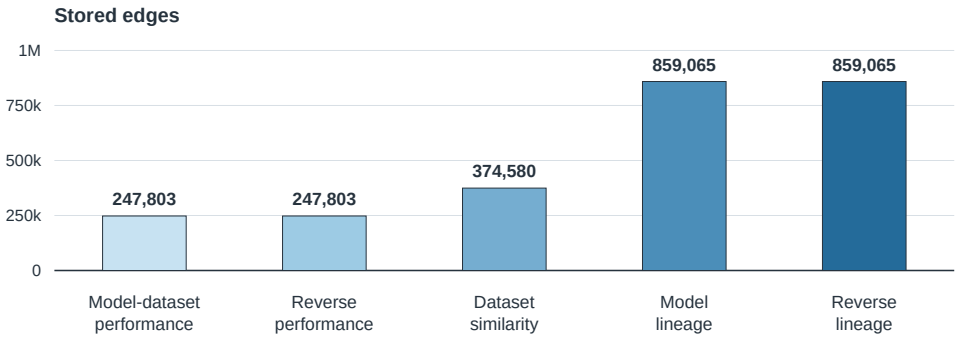

Figure 5: Stored directed edges by graph relation. Reverse directions are separate message relations. The five bars sum to 2,588,316 edges; the total is not itself a relation.

B.3 Training Configuration and Evidence Visibility

Fixed training configuration. The encoder uses one relation-specific GraphSAGE layer with 128-dimensional output. Relation gates are initialized to one. Adam optimization runs for 25 epochs at learning rate 0.01 with supervision batches of 1,024. Ordinary and lineage message-edge dropout rates are 0.30 and 0.05. The ranking, model-contrast, and lake-wide loss terms have equal weight; their temperatures τ_{rank}, τ_{contrast}, and τ_{global} are 0.10, 0.20, and 0.10, respectively. Model contrast draws 256 batch models per anchor, while the global objective draws 256 negatives per query. The final checkpoint is used; MSE, uniformity regularization, separate projection heads, and early stopping are disabled. This paragraph records the configuration, not an ablation.

Evidence visibility. Dataset roots are assigned to training, validation, and test, targeting approximately 70/10/20% of performance edges. Let the forward edge sets be $E_{\text{tr}}, E_{\text{val}}, E_{\text{te}}$. A disjoint 30% of E_{tr} is reserved as supervision E_{sup}. The message graphs expose these performance edges:

$$E^{\text{msg}}_{\text{train}} = E_{\text{tr}} \setminus E_{\text{sup}}, \quad E^{\text{msg}}_{\text{val}} = E_{\text{tr}}, \quad E^{\text{msg}}_{\text{test}} = E_{\text{tr}} \cup E_{\text{val}}. \tag{6}$$

Reverse performance edges follow the same visibility. Root split seeds are 0, 1, and 2, with initialization seed fixed at 0.

B.4 Historical Evidence-Based Reranker

For task t and model m, let n_{tm} denote the number of historical evidence values and A_{tm} their sum over eligible training and validation roots. We use a shrinkage prior with strength 5 and prior mean 0.5:

$$p_t(m) = \begin{cases} \dfrac{A_{tm} + 5 \cdot 0.5}{n_{tm} + 5}, & n_{tm} > 0, \\ 0, & n_{tm} = 0, \end{cases} \qquad r(d, m) = \frac{s_\theta(d, m) + 1}{2} + p_t(m) \tag{7}$$

Here $(s_\theta(d, m) + 1)/2$ maps the cosine-style retrieval score to $[0, 1]$, while $p_t(m)$ provides the historical task-specific bonus. For unseen task–model pairs, $p_t(m) = 0$.

B.5 Relation-Aware Graph Retrieval Encoder

Type-specific affine maps project both node types to $h_v^{(0)} \in \mathbb{R}^{128}$. For each relation r, let $\mathcal{N}_r(v)$ be the neighbors of node v under that relation, and let $\mathcal{R}(v)$ be the relation types whose destination is the type of v.

One relation-specific GraphSAGE layer (Hamilton et al., 2017) first averages these neighbors, then combines their features with the node's own features using parameters for that relation:

$$\mu_{v,r} = \frac{\sum_{u \in \mathcal{N}_r(v)} h_u^{(0)}}{\max\{1, |\mathcal{N}_r(v)|\}}, \qquad h_v^{(1)} = \sum_{r \in \mathcal{R}(v)} g_r \big(W_{s,r} h_v^{(0)} + W_{n,r} \mu_{v,r} + b_r \big),$$
$$z_v = \frac{W_O h_v^{(1)} + b_O}{\| W_O h_v^{(1)} + b_O \|_2}, \qquad s_\theta(d, m) = z_d^\top z_m. \tag{8}$$

Here $\mu_{v,r}$ is the average neighbor vector for relation r and is zero when $\mathcal{N}_r(v)$ is empty. The sum for $h_v^{(1)}$ includes only relations in $\mathcal{R}(v)$; $W_{s,r}$ and $W_{n,r}$ are the relation-specific self and neighbor projections, and g_r is a learned scalar gate.

C Supplementary Evaluation Details

This section expands the protocol in Section 4.1 and the measurements in Section 4.2. It records the interpretation of Figure 3; it does not report additional comparison runs or ablations.

C.1 Held-Out Query Eligibility

Eligible dataset–task queries have a known metric direction or a curated oriented source, exclude reinforcement-learning tasks and placeholder dataset names, and have at least three nonconstant observed held-out candidate values. The query counts in Section 4.1 are observations across three root-aware splits; the same node may occur in more than one split. All queried nodes already exist in the frozen graph, so this is a transductive historical-recovery evaluation, not a test of new-dataset ingestion.

C.2 Additional Recovery Metrics

Figure 3(d) shows several readouts of the same returned model lists. `gold@1` requires the observed best model to rank first; `top3@10` accepts any of the three best observed models among the returned ten; `gold-gap@10` accepts an observed model within 0.01 of the best oriented performance. The root-balanced measure first averages queries within each dataset root and then averages roots. The median reranked gold position, conditional on gold entering the candidate pool, is 4, 5, and 4 across the splits. The frozen candidate order resolves ties in the observed-best label; the same fixed model-ID order resolves ties in diagnostic scoring.

C.3 Approximate-Search Calibration

The index uses inner-product search with HNSW construction parameters $M = 32$ and `ef_construction`=200, using eight construction threads. Search breadth is selected from 1k, 1,500, 2,000, 3,000, and 5,000: the first setting with mean top-1k neighbor-ID recall of at least 0.99 against exact dense search is retained. This calibration does not read gold labels. The selected `ef_search` values for the three splits are 1k, 1,500, and 1k; the corresponding neighbor-ID recall values are 0.9944, 0.9963, and 0.9928. The task-prior tables are built from training and validation roots only; the test root and its siblings are excluded. This calibration checks search fidelity, not the ranking benefit of the prior or of graph learning.

C.4 Timing and Resource Accounting

The formal evaluation ran on Linux with `hnswlib` 0.8.0. HNSW search and prior fusion used the host CPU; exact diagnostic vector scoring used an NVIDIA L40S. Each split has one timed query pass using one HNSW query thread after up to 50 warm-up queries. The index was built separately with eight threads; its recorded build interval includes initialization, insertion, and serialization.

Figure 3(f) separates search from prior lookup, fusion, and ordering. The total is timed per query before percentiles are computed; component percentiles must not be added to reconstruct a total percentile. All times exclude query encoding, loading, and network overhead.

Figure 3(g–h) uses separate zero-based scales for training and shorter intervals. The two exact references share a scan, and the recorded evaluation interval already includes index construction; these bars are not additive pipeline stages. Each HNSW index occupies about 2.215 GiB. Training, vector export, and formal evaluation ran on different hosts, so the intervals do not form one same-machine runtime comparison.

Figure 3(i) separates training and evaluation resources. Process RSS high-water marks and PyTorch GPU allocated-memory peaks are different measures, not additive components. The RSS highs can include allocations made earlier by the process.