A Comprehensive Analysis of Dynamic Tiered BM25 Indexing and Query Routing using MSMARCO

Aaron Bengochea, Timothy Cao
(ab6503, tc4088)@nyu.edu
CS 6913: Web Search Engines – Fall 2025
NYU Tandon School of Engineering

December 21, 2025

Abstract

We conduct a comprehensive study using a production inspired pipeline for dynamic BM25 indexing with tiered partitions (T1/T2) and delta shards that absorb freshly ingested documents throughout the day. Documents are assigned to tiers via static BM25 scores derived from query-term frequencies, and deltas are periodically rolled into base indexes when thresholds are exceeded. At query time we overfetch from tiered and delta shards, rescore with global statistics, and we use a learned query router to decide between searching only T1 or both T1 and T2 in parralel. We compare tiered retrieval against a non-tiered BM25 baseline built on the same corpus, and evaluate multiple learned query routing models with varying classification routing thresholds to quantify quality versus cost trade-offs.

1 Learned Dynamic Tiered BM25 Indexing

1.1 Corpus Split and Working Set

We separate the corpus into a *train* subset and a *working* subset to decouple model development from evaluation and deployment. The train subset is used exclusively for computing static scores, deriving tier labels, extracting document side features, and training the tiering model. The working subset is reserved for emulating production tiered indexes and for all reported evaluation results. This separation prevents leakage of evaluation documents into training and mirrors a realistic pipeline where new content arrives after model training.

To construct the working subset, we begin from the full `collection.tsv` and ensure that every document ID appearing in the evaluation qrels (dev/eval1/eval2) is included. We first assemble the set of all relevant doc IDs from the qrels files and force-include them in the working subset. The remaining documents are split such that approximately 70% form the train subset and 30% form the working subset. For convenience and reproducibility, we persist the list of working doc IDs and generate the filtered collection files (`collection_train.tsv`, `collection_work.tsv`). This guarantees that any judged relevant document is present in the working indexes used for evaluation, while the training pipeline only sees the train subset.

By enforcing that all evaluation documents live in the working set, both the non-tiered baseline and the tiered indexes are built over the same evaluation corpus, and no evaluation document is inadvertently excluded.

1.2 Baseline BM25 Build

We build a single, non-tiered BM25 index over the working corpus to serve as the reference baseline. All BM25 builds (baseline and tiered) share identical preprocessing and scoring parameters to isolate the effect of tiering rather than parameter tuning. Preprocessing follows the standard pipeline: tokenization, lower-casing, stopword handling, and global statistics computations. We fix the BM25 hyperparameters for all indexes:

$$k_1 = 1.2 \qquad b = 0.75$$

During indexing we compute and persist collection-wide statistics: the number of documents N, average document length $avgdl$, and the lexicon with document frequencies. These statistics are reused by the tiered builds for consistent rescoring. By keeping the BM25 configuration constant across the baseline and all tiered indexes (T1, T2, and their deltas), any observed differences in retrieval quality are attributable to the tiering strategy and routing decisions, not to BM25 parameter drift.

1.3 Static Scoring and Tier Labels

We assign documents to tiers using a static relevance signal computed once over the working corpus. The procedure has three components: (i) building a query-term frequency (QTF) prior from historical queries, (ii) computing a static BM25 score for every document, and (iii) ranking and labeling documents according to a fixed tier ratio.

1.3.1 Query-Term Frequency Prior

We aggregate term counts over the full query log to obtain a prior importance weight for each term. The query log is the union of the development and evaluation query sets (`queries.dev.tsv` and `queries.eval.tsv`), combined as `queries.all.tsv`. After preprocessing (tokenization, lowercasing, stopword handling), for each query q with tokens $\{t\}$ we accumulate

$$\text{QTF}(t) \leftarrow \text{QTF}(t) + 1.$$

This produces a dictionary of term frequencies over the query distribution; rare terms can optionally be filtered if desired.

1.3.2 Static BM25 Score

Given the QTF prior, we compute a single static score per document as a weighted sum of BM25 term contributions. For document d with terms $t \in d$,

$$\text{Static}(d) = \sum_{t \in d} \text{QTF}(t) \cdot w(t, d)$$

where the BM25 term weight is

$$w(t, d) = \text{idf}(t) \cdot \frac{\text{tf}_{t,d} \cdot (k_1 + 1)}{\text{tf}_{t,d} + k_1 \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)}.$$

Here k_1 and b are fixed to the same values used in all BM25 indexes,

$$k_1 = 1.2 \qquad b = 0.75,$$

and $avgdl$ is the average document length from the working corpus. We iterate over the lexicon and postings once to accumulate $\text{Static}(d)$ for all documents.

1.3.3 Normalization and Labeling

The static scores are min–max normalized to $[0, 1]$,

$$\text{norm}(d) = \frac{\text{Static}(d) - \min}{\max - \min},$$

and documents are ranked by $\text{norm}(d)$ descending. A fixed tier ratio determines the split:

$$\text{Tier 1: } 40\% \qquad \text{Tier 2: } 60\%$$

The top fraction of documents are labeled Tier 1 ($y = 1$), and the remainder Tier 2 ($y = 0$). The resulting label files and score artifacts are persisted and used to build the tiered indexes and to train the document tiering model.

1.4 Tiering Model Choice (XGBoost)

1.4.1 Dynamic Document Feature Extraction

At ingestion time we can only rely on document-side signals; no query is available yet. We therefore derive features that are inexpensive to compute per document and capture length, rarity, and distributional structure:

- Static score and its log-transform: $\text{Static}(d)$ and $\log(1 + \text{Static}(d))$ carry the global relevance prior described above.
- Length features: raw document length $|d|$ and $\log(1 + |d|)$ to normalize scale.
- Rarity features: mean, max, and standard deviation of $\text{idf}(t)$ over terms in d (using the working-corpus lexicon); high values indicate rarer content.
- Lexical diversity: unique term count and the entropy of term-frequency distribution,

  $$\text{entropy}(d) = -\sum_i p_i \log p_i, \qquad p_i = \frac{\text{tf}_i}{\sum_j \text{tf}_j},$$

  measuring how concentrated or diffused the document vocabulary is.

For clarity, the full feature set used for tiering inference is:

- $\text{Static}(d)$
- $\log(1 + \text{Static}(d))$
- $|d|$
- $\log(1 + |d|)$
- mean of $\text{idf}(t)$ over $t \in d$
- max of $\text{idf}(t)$ over $t \in d$
- standard deviation of $\text{idf}(t)$ over $t \in d$
- unique term count
- $\text{entropy}(d)$ as defined above

All features are computed once per document from the working corpus statistics, making them suitable for a dynamic pipeline that continuously ingests new documents to be tiered dynamically.

1.4.2 Model Parameters

We use gradient-boosted decision trees (XGBoost) to learn a tiering classifier from these document features and static labels. XGBoost naturally handles heterogeneous feature scales, captures nonlinear interactions without manual feature engineering, and supports class-imbalance handling via `scale_pos_weight`. Training is performed on the train subset with a held-out validation split and early stopping. Core hyperparameters are fixed for consistency:

$$learing\_rate\_\eta = 0.05$$

$$\begin{aligned}
\text{n\_estimators} &= 800 \\
\text{max\_depth} &= 6 \\
\text{subsample} &= 0.8 \\
\text{colsample\_bytree} &= 0.8 \\
\text{scale\_pos\_weight} &= \frac{\#\text{T2}}{\#\text{T1}} \\
\text{early\_stopping\_rounds} &= 50 \\
\text{threshold} &= \tau = 40\% \quad \text{(chosen on validation to meet the desired tier1 ratio)}
\end{aligned}$$

We monitor validation area under the Receiver Operating Characteristic (ROC) curve and select a probability threshold τ (stored alongside the model) to match the desired tier ratio at inference. The resulting model and feature list are persisted for use during ingestion and can be used for document tiering inference without the need for GPU compute.

1.5 Tiered Index Dynamic Construction

1.5.1 Dynamic Ingestion and Delta Management

To avoid prior bias, we do not preload labels. Instead, the trained tiering model infers tier assignments dynamically over the working subset using a multiprocessing pipeline. The working subset is split into two partitions: an *init* split and a *delta* split. We first ingest the init split; as documents are inferred and routed to Tier 1 or Tier 2 deltas, the deltas rapidly exceed their thresholds, forcing a roll and rebuild into the base Tier 1 and Tier 2 indexes. We then ingest the delta split so that a non-trivial number of documents remain resident in Tier 1_delta and Tier 2_delta, preserving consistency for later analysis that includes deltas.

During ingestion, each worker batches documents, computes their features, obtains tier probabilities from the XGBoost model, and appends each document (docID, text) to the inferred tier's delta TSV. If a delta's line count exceeds its configured limit, the delta contents are flushed and merged into the corresponding base index, if delta thresholds are not crossed then we simply rebuild the smaller delta indexes which are can be built much faster due to their smaller size. This ensures that all documents in the tiered/delta indexes were assigned by the learned model under the same configuration, mirroring a live system that continuously ingests new content.

1.5.2 Index Layout and Rollovers

We materialize four BM25 indexes: two base tiers (Tier 1 and Tier 2) and two delta tiers (Tier 1_delta, Tier 2_delta) that absorb newly ingested documents. Each index is built with the same preprocessing and BM25 parameters as the baseline ($k_1 = 1.2$, $b = 0.75$), ensuring comparability.

The working set contains approximatly 2.6M documents, split into an *init* partition of roughly 2.1M documents and a *delta* partition of roughly 5̃00k documents. After model-driven inference and ingestion, the final tiered indexes contain about 8̃60k documents in Tier 1, 1̃.3M in Tier 2, 2̃00k in Tier 1_delta, and 3̃00k in Tier 2_delta. These ratios align with the intended 40/60 split used during training, indicating that the classifier's routing bias matches the target tier proportions.

Delta tiers are maintained as append-only inverted files fed by ingestion. We set rollover thresholds that trigger consolidation of delta content into the corresponding base tier:

$$\text{Tier 1 delta threshold: } 400{,}000$$

$$\text{Tier 2 delta threshold: } 1{,}000{,}000$$

When a delta exceeds its threshold, its postings are merged into the base tier, and the delta is cleared and rebuilt fresh. This keeps base indexes compact and stable, while deltas remain small and fast to update. All indexes persist collection statistics and lexicons so that query-time rescoring can use consistent global parameters. The layout mirrors a production environment: stable base tiers for high-throughput querying, plus lightweight deltas that capture recently ingested documents until periodic roll-in.

1.6 BM25 Tiered Retrieval

1.6.1 Query Plan and Candidate Gathering

At query time we execute a two-level plan: (i) decide whether to search only Tier 1 (and Tier 1_delta) or both tiers (and both deltas), and (ii) gather candidates from each selected index. In the absence of a query router, all queries search Tier 1, Tier 2, Tier 1_delta, and Tier 2_delta. When a router is enabled, the router's decision controls whether Tier 2 and its delta are searched.

1.6.2 Overfetching and Rescoring

To mitigate the approximation introduced by the splitting of documents over tiers, we overfetch from each index before merging. Given a target K, we fetch αK per index with an overfetch factor $\alpha > 1$ (default $\alpha = 2$). All candidates are then rescored using a consistent BM25 formulation with global statistics from the working corpus (N, $avgdl$, $k_1 = 1.2$, $b = 0.75$):

$$w(t,d) = \text{idf}(t) \cdot \frac{\text{tf}_{t,d} \cdot (k_1 + 1)}{\text{tf}_{t,d} + k_1 \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)}.$$

This rescoring step enforces a single scoring scale across base and delta indexes and across tiers, ensuring that the final ranking is comparable to a monolithic BM25 index.

1.6.3 Merging and Deduplication

Candidates from all consulted indexes are pooled and sorted by the rescored BM25 score. The final top-K list is emitted after this global merge. This merge-and-rescore pattern is essential for quality parity with a single-index baseline while preserving the operational benefits of tiering.

1.6.4 Parameter Choices and Performance

The overfetch factor α, the BM25 hyperparameters (k_1, b), and the use of global $N/avgdl$ are held constant across experiments to isolate the impact of tiering. Overfetch and rescoring add compute overhead, but by keeping deltas small and using consistent parameters we maintain latency within practical bounds while recovering recall lost to partitioning. These retrieval settings are the same ones used in the reported evaluations against the non-tiered baseline.

2 Learned Query Routing

2.1 Methodology

2.1.1 Motivation and Constraints

In a tiered retrieval system, the goal of query routing is to determine whether a query can be answered sufficiently using a smaller Tier-1 index or whether it should fall through to additional tiers. While this problem can naturally be framed as supervised classification, no ground-truth routing labels exist in practice.

Moreover, the MS-MARCO training relevance data presents an additional challenge. The available qrels are extremely sparse, often containing only a single judged relevant document per query. This sparsity makes traditional per-query effectiveness metrics unreliable as direct labeling signals.

These constraints strongly influenced our labeling strategy and ultimately motivated a rule-based approach grounded in judged relevance rather than metric comparisons.

2.1.2 Initial Consideration: Metric-Based Labeling (Abandoned)

Our initial approach was to define routing labels based on per-query effectiveness differences between Tier-1 and full retrieval. For example, if Tier-1 achieved MRR@10 or Recall@100 within some small ϵ of the full system, the query would be routed to Tier-1 only, otherwise, it would fall through.

However, with only one (or very few) judged documents per query, such metrics become unstable. MRR@10 effectively collapses to a binary signal, while Recall@100 becomes dominated by the presence or absence of a single document. In this regime, small rank differences can cause large metric swings that are unrelated to true query difficulty.

As a result, metric-based labeling was deemed too noisy to produce reliable supervision under sparse relevance judgments.

2.1.3 Primary Labeling Rule: Judged Hit in Top-K

To address qrels sparsity, we adopted a hit-based labeling strategy grounded directly in judged relevance.

For each training query, we retrieve top-k documents using:

- Tier-1 only retrieval
- Full retrieval (Tier-1 and Tier-2)

Labels are then assigned as follows:

- **Label 0 (Tier-1 sufficient)**: at least one judged relevant document appears in the Tier-1 top-k.
- **Label 1 (Fall through)**: no judged relevant document appears in Tier-1 top-k, but at least one appears in the full top-k.

- **Dropped**: no judged relevant document appears even in the full top-k.

This rule has several important properties. It is robust to sparse judgments, requiring only a single judged hit. It directly encodes the routing decision we care about (whether Tier-1 already retrieves something relevant) and avoids reliance on unstable per-query metric estimates.

This hit-based rule forms the core supervision signal for query routing.

2.1.4 Secondary Constraint: Pseudo-Recall Thresholding

While the hit-based rule is robust, it can be overly permissive. A query may retrieve a single judged document in Tier-1 while missing many other potentially relevant documents that are retrieved by the full system.

To address this, we introduce a pseudo-recall constraint as a secondary condition. For queries where Tier-1 retrieves at least one judged document, we compute:

$$\text{pseudo-recall} = \frac{|\text{Tier-1 top-}k \cap \text{Full top-}k|}{|\text{Full top-}k|}$$

Here, the full system's top-k is treated as a proxy for the set of potentially relevant documents.
A query is labeled as Tier-1 sufficient only if both conditions hold:

1. A judged hit appears in Tier-1.
2. The pseudo-recall exceeds a threshold t.

If the pseudo-recall falls below t, the query is labeled as fall-through, even though Tier-1 retrieved a judged document.

This design penalizes cases where Tier-1 retrieves a judged hit but diverges substantially from full retrieval, while also introducing a tunable parameter that allows controlled exploration of effectiveness and efficiency trade-offs.

2.1.5 Thresholds and Label Variants

Rather than selecting a single pseudo-recall threshold, we train separate routing models using thresholds:

$$t \in \{0.0, 0.1, 0.2, \ldots, 0.9\}$$

The case $t = 0.0$ corresponds to the pure hit-based rule, where no pseudo-recall constraint is enforced. Higher thresholds encourage increasingly conservative routing, requiring Tier-1 to recover a larger fraction of the documents retrieved by the full system.

Training multiple models allows us to study routing behavior across operating points, avoid baking a single policy into the model, and evaluate routing sensitivity without retraining indexes. In effect, the threshold becomes a tunable control knob for routing aggressiveness.

2.1.6 Query Features

We restrict ourselves to query-only features, ensuring that routing decisions can be made dynamically at inference time, before any retrieval over postings lists occurs. This design allows each incoming query to be routed independently and efficiently, without requiring additional index access or offline preprocessing.

The features fall into two categories.

1. Structural query features capture query verbosity and redundancy:

- Number of terms
- Number of characters
- Number of unique terms
- Fraction of unique terms
- Average term length

2. Collection-level IDF statistics capture term selectivity:

- Maximum IDF
- Minimum IDF
- Mean IDF
- Standard deviation of IDF

Intuitively, short queries with rare, high-IDF terms are more likely to be answered sufficiently by a smaller index, whereas longer or more generic queries benefit from broader coverage. All features are computable using only the query text and collection-level statistics, enabling dynamic per-query routing decisions in a latency-critical deployment setting.

Importantly, feature definitions are fixed and identical across all models, ensuring that differences in routing behavior arise solely from thresholding and training data, rather than feature drift or inference-time variability.

2.1.7 Model Choice

We use a lightweight linear classifier (logistic regression) for routing.

The routing decision is binary and low-dimensional, making linear models a natural fit. Logistic regression is fast, interpretable, and easy to deploy, which is important for a component that sits on the critical path of query processing.

More complex models such as gradient boosting were considered but rejected to avoid unnecessary complexity and latency. Training separate classifiers for each threshold allows the decision boundary to adapt naturally to different routing aggressiveness levels, rather than forcing a single model to approximate multiple operating points.

2.2 Implementation

2.2.1 Index and Dataset Partitioning

During the index tiering phase of the project, the MS-MARCO document collection was split into:

- `collection_train.tsv`, which was used exclusively to train the index tiering model.
- `collection_work.tsv`, which contains all documents referenced by evaluation qrels plus additional randomly sampled documents.

All tiered indexes used for query routing are built from `collection_work.tsv`, which represents the deployed system's document universe. This separation ensures that index tiering decisions are learned from training data, while routing decisions operate on the same indexes used at evaluation time.

2.2.2 Training Data Selection

To train the query routing model, we start from MS-MARCO's `queries.train.tsv` and `qrels.train.tsv`. Because the corpus was artificially partitioned, not all judged documents appear in the working collection. We therefore filter training queries to those whose judged documents exist in `collection_work.tsv`.

To keep retrieval tractable, we randomly sample 5,000 queries from the filtered set, a strategy approved by the course instructor.

2.2.3 Training Pipeline

For each sampled training query, we:

1. Retrieve top-k using Tier-1.
2. Retrieve top-k using full retrieval.
3. Generate labels using the hit-based rule and pseudo-recall threshold.
4. Extract query features.
5. Repeat for each threshold t.

This produces 10 labeled datasets, one per threshold, from which we train 10 independent routing models.

2.2.4 Inference and Evaluation

At inference time, each incoming query is featurized and passed through a selected routing model, which dynamically determines whether the query can be answered using Tier-1 alone or should fall through to additional tiers. This decision is made independently for each query and does not require any prior retrieval or access to postings lists. Retrieval is then executed accordingly using the tiered indexes.

We evaluate routed retrieval on eval1 and eval2 using all thresholds, and on the dev set using a subset of thresholds due to its larger size. Metrics are reported both overall and stratified by query length.

This structure mirrors a realistic deployment scenario, where routing decisions must be made online and adaptively on a per-query basis.

3 Detailed Analysis

3.1 Experimental Setup

We evaluate three query sets from MSMARCO: dev (binary relevance), eval1, and eval2 (graded relevance). Each query set is run against two controls: (i) a non-tiered BM25 index built on the working corpus, and (ii) a tiered BM25 configuration that forces visitation of all tiers and their deltas (Tier 1, Tier 2, Tier 1_delta, Tier 2_delta). These controls anchor quality without query routing.

On top of the tiered setup, we apply learned query routing using ten router variants with different recall thresholds (t00–t90; see Section "Learned Query Routing" for details). The router decides whether to search only Tier 1 (and Tier 1_delta) or both tiers (and both deltas). All runs use identical BM25 parameters ($k_1 = 1.2$, $b = 0.75$), overfetching, and global rescoring to ensure fair comparison.

Metrics: for the dev set we report MRR@10, Recall@100, and MAP; for eval1 and eval2 we report MRR@10, Recall@100, NDCG@10, and NDCG@100.

3.2 Query Routing Impact

The learned routers (T00–T90) apply progressively stricter recall thresholds derived from our training process. Lower-numbered models such as T00 are more permissive and classify more queries as Tier 1 only, while higher-numbered models such asT90 are more conservative and route most queries to Tier 1&Tier 2. The tables below summarize, for each query set, how many queries were routed to Tier 1 only versus both tiers as a function of the chosen router.

Model	**T1 Only (y=0)**	**T1&T2 (y=1)**
T00	22	21
T10	20	23
T20	19	24
T30	17	26
T40	15	28
T50	13	30
T60	11	32
T70	10	33
T80	8	35
T90	6	37

Table 1: Routing decisions on eval1 (total queries = 43).

Model	**T1 Only (y=0)**	**T1&T2 (y=1)**
T00	33	21
T10	31	23
T20	29	25
T30	28	26
T40	26	28
T50	23	31
T60	21	33
T70	18	36
T80	13	41
T90	6	48

Table 2: Routing decisions on eval2 (total queries = 54).

Model	**T1 Only (y=0)**	**T1&T2 (y=1)**
T00	548	452
T10	516	484
T20	477	523
T30	445	555
T40	399	601
T50	359	641
T60	315	685
T70	282	718
T80	244	756
T90	206	794

Table 3: Routing decisions on dev (total queries = 1000).

We next apply these routers at varying thresholds across all evaluation sets (dev, eval1, eval2). For each evaluation, we compare against two controls: *all*, which forces every query to search both tiers and their deltas, and *untiered*, which queries a single non-tiered index built on the same working corpus. The tables below report retrieval quality after routing for each thresholded model and the two controls.

Threshold	MRR@10	Recall@100	NDCG@10	NDCG@100
t00	0.7377	0.4288	0.4139	0.4489
t10	0.7398	0.4311	0.4178	0.4521
t20	0.7421	0.4345	0.4201	0.4567
t30	0.7455	0.4387	0.4256	0.4617
t40	0.7483	0.4434	0.4298	0.4647
t50	0.7518	0.4474	0.4323	0.4689
t60	0.7543	0.4472	0.4376	0.4723
t70	0.7573	0.4501	0.4389	0.4763
t80	0.7593	0.4543	0.4405	0.4792
t90	**0.7610**	**0.4596**	**0.4423**	**0.4884**
untiered	0.7672	0.4602	0.4348	0.4881
all	0.7632	0.4641	0.4479	0.4938

Table 4: Eval1. Best scores per metric are shown in bold.

For eval1, the most permissive router (T90) delivers the strongest routed results among the learned models, but it still trails the untiered and all baselines. This is expected: routing some queries to Tier 1 only introduces early-termination risk that can drop recall. Overfetching in the tiered setup lets the all configuration slightly surpass the untiered baseline, though the gain is small; latency remains acceptable because Tier 1 and Tier 2 (and their deltas) are searched in parallel. Notably, the spread between untiered/all and mid-range routers (e.g., T70–T80) across MRR@10, Recall@100, NDCG@10, and NDCG@100 is modest, suggesting that a router like T70 could be paired with larger first-stage such as top-k 1000 and BM25 early termination to feed deeper reranking cascades with minimal quality loss while reducing overall latency.

Threshold	MRR@10	Recall@100	NDCG@10	NDCG@100
t00	0.7124	0.3386	0.4203	0.3777
t10	0.7143	0.3392	0.4227	0.3798
t20	0.7166	0.3413	0.4260	0.3810
t30	0.7177	0.3423	0.4278	0.3822
t40	0.7196	0.3454	0.4298	0.3832
t50	0.7206	0.3501	0.4323	0.3887
t60	0.7252	0.3652	0.4378	0.3911
t70	0.7399	0.3823	0.4469	0.4075
t80	0.7477	0.4032	0.4584	0.4210
t90	**0.7682**	**0.4274**	**0.4611**	**0.4431**
untiered	0.7704	0.4549	0.4775	0.4676
all	0.7868	0.4603	0.4805	0.4701

Table 5: Eval2. Best scores per metric are shown in bold.

For eval2, we observe the same pattern: more permissive routers improve quality, with T90 leading the routed variants yet still trailing the untiered and all baselines. The recall gap is slightly steeper on eval2, indicating this query set is harder, but the qualitative takeaway is unchanged. Mid-range routers (T70–T80) remain close to the baselines, implying a viable latency/quality trade-off when paired with larger first-stage top-k and early termination prior to downstream reranking in order to reduce initial candidate generation latency.

Threshold	**MRR@10**	**Recall@100**	**MAP**
t00	0.2273	0.5266	0.2204
t10	0.2292	0.5291	0.2232
t20	0.2309	0.5337	0.2266
t30	0.2317	0.5387	0.2287
t40	0.2354	0.5433	0.2301
t50	0.2382	0.5532	0.2331
t60	0.2411	0.5673	0.2369
t70	0.2456	0.5845	0.2392
t80	0.2481	0.5903	0.2403
t90	**0.2505**	**0.6078**	**0.2449**
untiered	0.2732	0.6954	0.2682
all	0.2856	0.7126	0.2806

Table 6: Dev. Best scores per metric are shown in bold.

For dev, the recall drop is steeper than in eval1 and eval2, driven by the larger query set (1,000 queries). Even so, absolute recall remains roughly 16–20% higher than the eval sets, suggesting that with larger first-stage top-k (e.g., 1000) we could reasonably adopt mid-range routers (T70–T80) as a default. Additional document/query expansion could further mitigate drift and lift MRR@10 and Recall@100. As with the other evals, this stage can serve as the initial candidate generator with a top-k 1000 in a multi-stage retrieval and reranking cascade, where later stages correct general ranking where our goal is to generate a final top-k 100.

3.3 Conclusions

We presented a production inspired dynamic BM25 tiering system with learned document tiering and query routing. Static scores derived from query-term frequencies produced a clean 40/60 Tier 1/Tier 2 split, and the XGBoost tiering model reproduced that bias when ingesting 2.6M working documents into base and delta tiers. Query routing experiments across dev, eval1, and eval2 show that mid-range routers (T70–T80) incur only modest quality loss relative to untiered or “all tiers” controls, while enabling early termination for roughly 20–35% of queries (T1-only decisions). This supports lower-latency top-k 1000 candidate generation with minimal recall loss, leaving later reranking stages to correct residual drift.

When combined with tiered HNSW retrieval in a cascade, the higher-quality dense candidates can compensate for the slight recall erosion from BM25 routing, yielding an overall pipeline that balances efficiency and effectiveness. Initial BM25 and HNSW top-k 1000 early candidate generation can be casdaded into complex reranking strategies such as Reciprocal Rank Fusion (RRF) which can then be applied to Bert based Bi-encoders to assure high quality top-k 100 outputs for our dynamic tiered retrival system.

3.4 Future Work

- Complete tiered HNSW and integrate joint routing so lexical and dense tiers supply a richer top-k candidate pool in parallel (tiers are already queried in parallel but different indexes can also be queried in parallel).
- Explore cascaded reranking that stacks RRF, bi-encoders, and cross-encoders; measure whether stronger cascades permit more aggressive BM25 early termination (potentially T50–T60) without hurting final quality.
- Calibrate query routers with cost-aware objectives and adaptive thresholds keyed to query difficulty signals.
- Broaden ablations on tier ratios, delta roll-in thresholds, and overfetch factors to refine the latency/quality frontier.
- Add memory- and cache-aware indexing policies for deltas and bases to reduce footprint while maintaining recall.
- Automate freshness and drift testing under continuous ingestion and roll-in schedules to harden the system for production workloads.
- Evaluate ensemble routing that jointly considers BM25 and HNSW signals to decide tier selection per query.