Efficient Hybrid Search for Music Discovery

CS6400-A Final Project Report (Group 21)

Vanshika Shah
vshah316@gatech.edu

Eric Shao
eric.shao@gatech.edu

Yangsheng Zhou
yzhou895@gatech.edu

1 INTRODUCTION

The music industry has seen rapid growth over the years and there has been a lot of scope in building an efficient and accurate search system that can take care of both semantic aspects of the audio and structured constraints of metadata. Traditional approaches have existed for performing hybrid search. These methods performed a vector similarity search followed by structured filtering. Such methods depended on simple pre-filtering or post-filtering pipelines which often suffered from scalability bottlenecks and wasted computation [2]. For this project we looked into improvements that could be made to search systems to improve performance and speed [1, 4].

In this project we evaluated a hybrid music search system that fused vector similarity over Spotify's 19-dimensional audio feature vectors with precise metadata constraints in a single query process. While hybrid retrieval is conceptually straightforward, naïve implementations introduce fundamental trade-offs between accuracy and efficiency. Pre-filtering applies metadata constraints first and then performs an exact search over the reduced dataset, this ensures correctness but scales poorly when filters are broad, since the system still operates over tens of thousands of vectors. Post-filtering, on the other hand, retrieves candidates using unfiltered vector search and applies constraints afterward; although faster, it frequently suffers from recall failures, especially when the ANN stage does not retrieve enough valid items to satisfy the metadata conditions. These limitations make both classical approaches unreliable or inefficient across different selectivity levels.

To overcome the issues in the classical approach, we developed a FAISS-based hybrid architecture that integrates stuctured filtering directly in to the vector search process through predicate pushdown. Our method does not retreive all the clusters and perform filtering afterwards, instead, it precomputes cluster-level metadata bitmaps and uses them to prune entire FAISS IVF lists that do not contain any valid tracks. This approach ensures that clusters containing valid items take part in vector search, and reduce unneccesary distance computations making sure that valid neighbours remain in the candidate set. Essentially, the system combines the selectivity-awaremness of pre-filtering with the efficiency of ANN traversal.

Our main contributions through this work are: (1) A hybrid retrieval pipeline that fuses semantic vector similarity with strict metadata constraints using cluster-level predicate pushdown. (2) An indexing framework including FAISS IVF-Flat for embeddings, and categorical and numeric metadata indexes for fast filtering. (3) An evaluation setup with an exact ground-truth search index, a diverse query workload, and baselines covering pre-filtering and post-filtering.(4) Analysis that shows our hybrid method does acheive lower latency than pre-filtering while maintaining high recall than post-filtering. Our results show that integrating metadata-aware pruning into the vector search path, helped develop an effective way to accelerate hybrid retrieval without compromising accuracy.

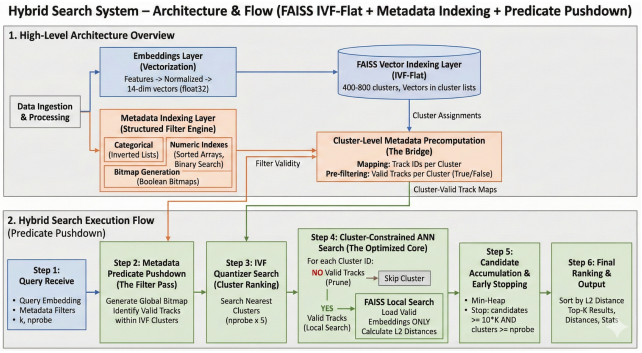

Figure 1: Architecture and flow

2 DESIGN AND IMPLEMENTATION

2.1 System Architecture Overview

Our hybrid search system integrates vector similarity search with metadata filtering through a novel predicate pushdown approach. Figure 1 illustrates the complete architecture, which consists of two main sections: the high-level architecture overview (top) and the hybrid search execution flow (bottom).

2.1.1 High-Level Architecture Overview. The system architecture comprises four major layers connected through a critical "bridge" component:

Embeddings Layer (Vectorization): Raw audio features from the Spotify dataset are normalized and transformed into 14-dimensional vectors stored as float32 arrays. This compact representation balances discriminative power with memory efficiency, enabling fast in-memory operations.

FAISS Vector Indexing Layer (IVF-Flat): The embeddings are organized using FAISS IVF-Flat structure with 400-800 clusters. Each cluster maintains a list of vectors, creating an inverted file structure that enables sub-linear search complexity. The index partitions the 170,000-track embedding space using k-means quantization.

Metadata Indexing Layer (Structured Filter Engine): This layer maintains two specialized index types for efficient metadata filtering:

- **Categorical Indexes (Inverted Lists):** Hash-based structures mapping discrete values (explicit flags, musical keys) to sets of track IDs
- **Numeric Indexes (Sorted Arrays, Binary Search):** Sorted value arrays with corresponding track indices enabling

logarithmic-time range queries on continuous features (year, tempo, duration, energy, danceability)
- **Bitmap Generation:** Boolean bitmaps created by combining filter conditions, representing valid/invalid tracks

Cluster-Level Metadata Precomputation ("The Bridge"): This is the critical innovation connecting vector and metadata indexes. For each IVF cluster, we precompute:

- **Mapping:** Complete list of track IDs belonging to each cluster
- **Pre-filtering:** Boolean flags indicating whether each cluster contains ANY valid tracks satisfying current metadata filters

This bridge component enables the two-level filtering hierarchy that distinguishes our approach from standard filtered vector search.

2.1.2 Hybrid Search Execution Flow (Predicate Pushdown). The query execution pipeline proceeds through six carefully orchestrated steps:

Step 1: Query Receive The system accepts three inputs: the query embedding (14-dim vector), metadata filter conditions (conjunctions of categorical and numeric constraints), and search parameters (K for top-K results, nprobe for cluster examination depth).

Step 2: Metadata Predicate Pushdown ("The Filter Pass") This step generates the global validity bitmap and identifies valid tracks within each IVF cluster:

- Apply all metadata filters to create a global boolean bitmap marking which tracks satisfy all predicates
- For each of the 400-800 IVF clusters, intersect the cluster's track membership with the global validity bitmap
- Store cluster-level validity flags (true/false) and the filtered track lists for valid clusters

This precomputation is the first level of our hierarchical filtering approach, enabling entire clusters to be pruned before any distance computations occur.

Step 3: IVF Quantizer Search (Cluster Ranking) Using the IVF quantizer (the trained k-means centroids), we identify the nearest clusters to the query vector:

- Search for nprobe × 5 nearest clusters (not just nprobe)
- This oversampling compensates for clusters that will be filtered out in Step 4
- Results in a ranked list of cluster IDs ordered by distance from query to cluster centroid

The 5× oversampling is a key adaptation ensuring sufficient valid clusters remain after filtering.

Step 4: Cluster-Constrained ANN Search ("The Optimized Core") This is where the two-level filtering hierarchy executes. For each cluster ID in the ranked list:

Cluster-Level Decision:

- Check if cluster has ANY valid tracks (using precomputed flags from Step 2)
- If **NO valid tracks**: Skip cluster entirely—no memory access, no distance computations
- If **YES valid tracks**: Proceed to track-level filtering

Track-Level Search (FAISS Local Search):

- Load embeddings ONLY for valid tracks within this cluster (not all cluster members)
- Calculate L2 distances between query and valid embeddings only
- Collect top candidates from this cluster (limited to prevent memory explosion)

This two-level approach exploits the IVF cluster structure: cluster-level pruning eliminates entire memory regions, while track-level filtering ensures we only compute distances for metadata-compliant tracks.

Step 5: Candidate Accumulation & Early Stopping Candidates from examined clusters are accumulated in a min-heap structure. Two stopping conditions enable early termination:

- **Sufficient candidates**: Total candidates $10 \times K$ (ensures diverse result pool)
- **Sufficient exploration**: Valid clusters examined nprobe (maintains recall guarantees)

Both conditions must be satisfied before stopping. This prevents premature termination when filters are highly selective.

Step 6: Final Ranking & Output All accumulated candidates are sorted by L2 distance in a single global ranking operation. The top-K results are extracted along with their distances and performance statistics (latency, number of clusters examined, number of distance computations).

2.2 Core Algorithm Analysis

2.2.1 The Two-Level Filtering Hierarchy: Our fundamental innovation applies metadata predicates hierarchically:

Cluster-Level Pruning: Before examining vectors, determine if clusters contain any valid tracks. Invalid clusters are skipped entirely, avoiding memory loads and distance computations. For selective filters (e.g., year=2020, tempo [120,140]), 50-80% of clusters can be pruned.

Track-Level Filtering: Within valid clusters, load and compute distances only for metadata-compliant tracks, never wasting computation on invalid vectors.

2.2.2 Adaptive Oversampling: Searching nprobe × 5 clusters (instead of exactly nprobe) compensates for filtered clusters, maintaining recall consistency across varying selectivities (1-80%).

2.3 Custom Components

(1) **MetadataFilter Class:** Encapsulates categorical inverted lists and numeric sorted arrays with unified filtering interface
(2) **Cluster-Level Metadata Precomputation (The Bridge):** Builds cluster_to_tracks mappings and computes cluster validity flags per query
(3) **HybridSearcher Class:** Orchestrates six-step execution with adaptive oversampling and two-level filtering
(4) **Query Workload Generator:** Creates 100+ queries with controlled selectivity (1-80%)
(5) **Evaluation Harness:** Automates nprobe tuning, method comparison, metrics collection, and visualization

3 EXPERIMENTAL SETUP

The experiments were executed on macOS using Python 3.9 and the following core libraries:

- **FAISS-CPU (1.10), (1.7+)** for vector search (IndexFlatL2, IVFFlat)
- **NumPy, Pandas, scikit-learn** for preprocessing, embedding construction, bitmaps and filtering
- **Matplotlib/Seaborn** for plotting and analysis.

All results were computed using single-threaded FAISS-CPU to ensure consistent comparisons across methods.

3.1 Dataset Characteristics and Query Workload

Dataset Characteristics. We use the Spotify Tracks Dataset [3] containing 170,653 vectors across 19 dimensions. We performed basic cleaning to remove extreme tempos and durations and finally were left with 169,776 tracks with both audio features and metadata fields:

- **Numerical audio features:** danceability, energy, valence, tempo, acousticness, instrumentalness, liveness, speechiness, loudness, popularity
- **Categorical metadata:** key, mode, explicit flag, artists
- **Temporal/numeric metadata:** year, duration_ms

After the basic cleaning, we generated a 14-dimensional embedding where we combined (1) Standardized audio features, (2) Weighted feature scaling (to better reflect musical similarity), (3) Lightweight categorical encoding (key, mode, explicit), and, (4) Scaled year feature. These steps give a 14-dimensional embedding that preserves the most musically meaningful structure in the data while avoiding redundancy across correlated audio features. Standardization ensures that all numerical features contribute proportionally to the L2 geometry, while weighted scaling emphasizes perceptually important attributes such as energy, and rhythmic characteristics. Lightweight categorical encodings (key, mode, explicit) allow the embedding to incorporate musical modality without inflating dimensionality, and scaling the year feature adds a temporal component that captures evolution over time.

Index Configuration. For vector search, we built two FAISS indexes as seen in Table 1:

- **IndexFlatL2**: Exact nearest neighbor search for ground truth computation
- **IndexIVFFlat**: Approximate nearest neighbor search with nlist = 412 clusters ($\sqrt{n}$), used for both baseline and hybrid methods

The system was configured with nprobe values ranging from 1 to 500 to analyze the accuracy-latency trade-off, with default nprobe = 20 for most experiments.

Query Workload. To evaluate our baseline and hybrid retrieval methods under realistic and diverse conditions, we construct a structured query workload that spans a wide range of metadata selectivities. Each query consists of two components: (1) a query track ID used for vector similarity, and (2) a metadata filter specifying structured constraints such as year ranges, explicitness, tempo ranges, or categorical values. Some examples can be seen in Table 3

The workload intentionally includes high-selectivity, low-selectivity, and mixed-selectivity cases. High-selectivity queries match only a small subset of tracks (e.g., specific artist, narrow year range), helping test scenarios where filtering early should be advantageous. Low-selectivity queries use broad numeric ranges or minimal filtering, highlighting conditions where post-filtering or approximate methods may be more efficient. Mixed-selectivity queries combine narrow and broad constraints, stressing whether the retrieval strategy prioritizes filters effectively.

By generating 100–150 queries covering multiple metadata dimensions, we ensure that evaluations of pre-filter, post-filter, and hybrid methods are consistent, repeatable, and representative of real usage patterns. The same query workload is shared across all experiments, enabling fair comparison of latency, recall, and selectivity-dependent performance.

3.2 Evaluation Metrics

We employed the following metrics to comprehensively evaluate system performance:

(1) **Recall@K**: Fraction of ground-truth nearest neighbors appearing in the top-K results, where ground truth is computed via brute-force search on filtered data

$$\text{Recall@K} = \frac{|\text{Retrieved} \cap \text{GroundTruth}|}{K}$$

(2) **Query Latency**: End-to-end execution time in milliseconds, measured from query submission to result delivery
(3) **Distance Computations**: Number of vector distance calculations performed during search, indicating computational efficiency
(4) **Selectivity Impact**: For each query, we compute the number of items surviving the structured filter. This helps interpret when post-filtering fails and when hybrid methods skip entire clusters successfully.

3.3 Baseline Implementations

We implemented and compared against two standard baseline approaches:

(1) **Pre-filtering Baseline**: This approach first applies all structured metadata constraints directly to the full dataset. Once filtering reduces the candidate set, we perform an exact L2 nearest-neighbor search over only the surviving vectors. Because distance computations are restricted to valid items, this method consistently achieves very high recall whenever at least K items pass the filter. In practice, this baseline effectively acts as our ground-truth reference for filtered queries, since it performs exhaustive search within the correct subset.
(2) **Post-filtering Baseline**: This approach prioritizes speed by retrieving a large unfiltered candidate pool (pool_size = 500) using a full IndexFlatL2 search before applying metadata filters. Invalid items are removed, and the remaining candidates are reranked with exact L2 distances to produce the top-K results. Note that our hybrid method uses an ANN-style IVF search with nprobe (results shown in Section 4.2), we keep the pool_size-based post-filtering baseline

to enable a consistent comparison against traditional approaches. This baseline achieves lower latency on broad queries but suffers noticeable recall degradation on selective filters. Both pool size and ANN style implementation give similar results for post-filtering as shown in section 4.

Both baselines were implemented from scratch to ensure fair comparison and were evaluated using the same query workload and ground truth computation. The hybrid methods (Predicate Pushdown and Intersect-then-Rank) were compared against these baselines across all metrics.

Table 1: FAISS Index Configuration Parameters

Parameter	Value	Description
Embedding Dimension	14	Reduced feature dimension
Total Vectors	169,776	Number of track embeddings
nlist	412	Number of IVF clusters
Index Type	IVF-Flat	Approximate nearest neighbor
Distance Metric	L2	Euclidean distance
Default nprobe	20	Clusters probed per query

4 EVALUATION

We have created the same embeddings and query workloads for both the the baseline and hybrid search implementation.

4.1 Baseline results

This section shows the evaluation for the Pre- and Post-Filter implementation on our dataset across the full query workload described earlier. Table 4 summarizes the average performance for k = 10 and Table 5 shows a sweep across multiple values of k ∈ {5, 10, 20, 50}.

Across all queries, pre-filtering achieves the highest (perfect) recall (1.0) since the brute force search is performed directly on the filtered subset, ensuring that all retrieved neighbors satisfy the metadata constraints. We treat this as the ground truth for our implementation. However, this comes at the cost of higher latency, an average of 9.3 ms, due to the overhead of computing distances on the filtered subset for every query.

In contrast, post-filtering shows significantly lower latency (approx. 2.09 ms) because the system first retrieves approximate nearest neighbors from the full FAISS index and applies metadata constraints only afterward. This results in an order-of-magnitude reduction in computation. However, because many high-similarity neighbors are discarded during post-filtering, the recall drops sharply to 0.53, confirming that post-filtering is unreliable when filters are selective or interact strongly with the embedding space.

Table 5 further illustrates how these patterns persist as Pre-filter recall remains consistently high (1.0), while post-filter recall stays around 0.46–0.55. Latencies show very little sensitivity to k which shows that the cost is driven primarily by per-query filtering and distance evaluation, rather than the returned top-K size.

Figure 2 shows that Pre-filter queries cluster at high recall but higher latency, while Post-filter queries cluster at low latency but highly variable recall. This verifies that the two baselines represent opposite ends of the accuracy–efficiency trade-off and establishes a clear performance gap that the hybrid method aims to close.

Figure 3 shows that Post-filter recall sharply degrades as filter selectivity increases. This verifies that Post-filter is unstable under broad filters because a large fraction of ANN-retrieved candidates fail metadata checks. The trend confirms that selectivity is a key bottleneck for Post-filter performance.

Table 2: Metadata Filter Types and Implementation

Filter Type	Examples	Index Structure	Lookup Complexity
Categorical	mode, explicit, key	Hash map (value → indices)	O(1)
Numeric Range	year, tempo, duration	Sorted arrays + binary search	O(log n)
Multi-value	artists (multiple)	Inverted index	O(k)
Combined	year + explicit + tempo	Bitmap intersection	O(m)

Table 3: Query Workload Distribution by Selectivity

Selectivity Type	Count	Avg. Filtered Tracks	Example Query
High (>20%)	94	~85,000	explicit=0, key=6
Medium (5–20%)	37	~25,000	year=[2007,2010]
Low (<5%)	19	~8,500	year=[2006,2007], explicit=0

Table 4: Summary of Baseline Performance Across All Queries for k=10

Method	Mean Recall@10	Mean Latency (ms)	Notes
Pre-Filter	1.0	13.715	high accuracy, poor efficiency
Post-Filter (pool_size=500)	0.525	2.097	high efficiency, poor accuracy

Table 5: Baseline Recall and Latency Across Different Values of K

K	Pre-R	Post-R	Pre-L (ms)	Post-L (ms)
5	1.0	0.546	12.510	1.907
10	1.0	0.524	13.715	2.097
20	1.0	0.498	11.679	1.86
50	1.0	0.456	11.2	1.763

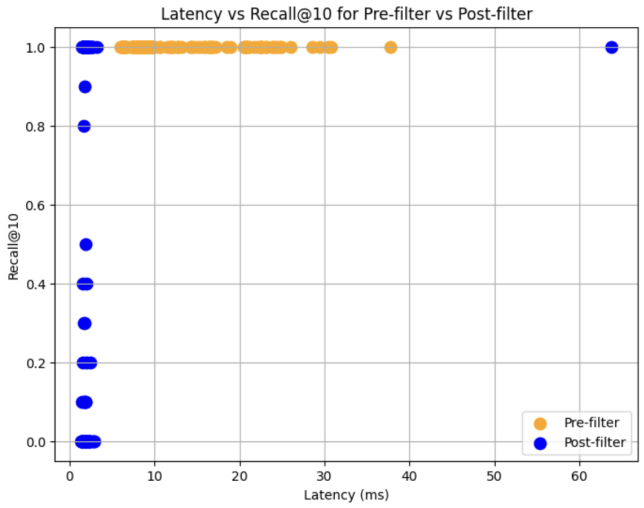

Figure 2: Latency vs Recall@10 for baseline Methods

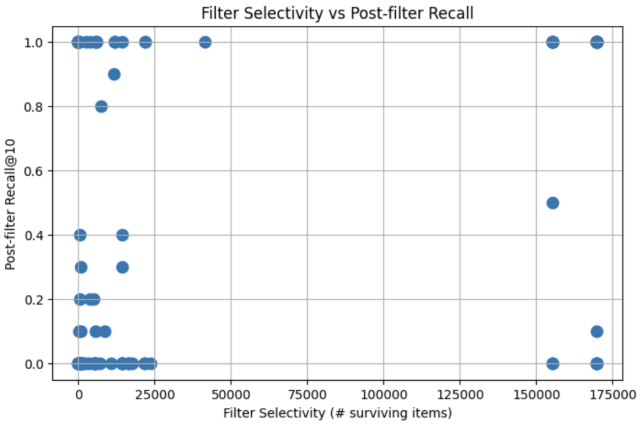

Figure 3: Filter Selectivity vs Post-filter Recall at k=10

Table 6: Summary of Baseline Performance Across All Queries for k=10

Method	Mean Recall@10	Mean Latency (ms)
Pre-Filter	1.0	11.14
Post-Filter	0.6.867	1.96
Hybrid-Method	0.885	7.91

4.2 Predicate Pushdown Results

4.2.1 Experimental Results. We evaluated three filtered vector search approaches across 150 queries with varying selectivity (1%-95%), categorized into low (<5%), medium (5-20%), and high (>20%) buckets Table 6 shows pre-filter achieves perfect recall through exhaustive search. Post-filter offers lowest latency but reduced recall. Hybrid balances both at 88% recall and 7.9ms latency.

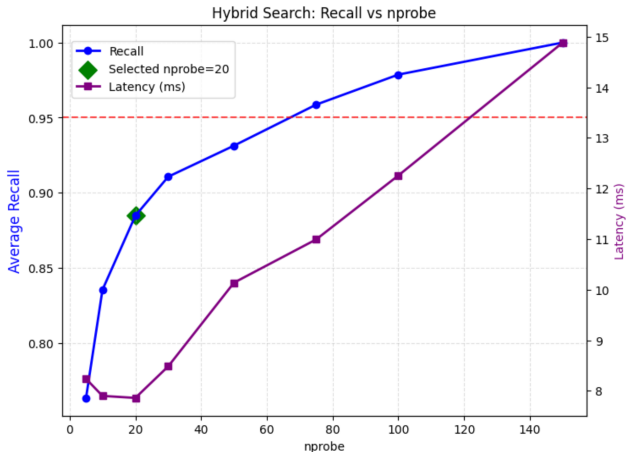

Figure 4: recall-latency tradeoff with different nprobe

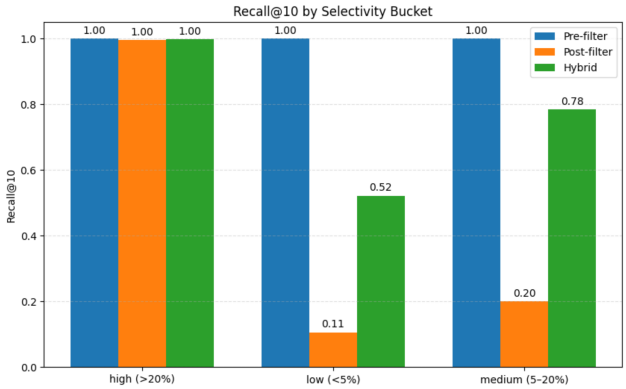

Figure 5: Recall by Selectivity bucket

Nprobe Tuning: Figure 4 shows the recall-latency tradeoff for varying nprobe. We selected **nprobe=20** (green diamond) achieving 88% recall at 8ms. Beyond this, recall improvements are marginal while latency nearly doubles—reaching 95% recall (red dashed line) requires nprobe75 with 50% latency penalty.

4.2.2 Performance by Selectivity. Figure 5 and Figure 6 show Performance by Selectivity Bucket.

4.2.3 Analysis.

Low Selectivity (<5%). Figure 5 (middle cluster) shows dramatic method differences:

- **Post-filter fails** (11% recall) because retrieving 100 candidates from 500 valid tracks misses 90% of results
- **Pre-filter excels** (4.4ms) as brute-force over 500 tracks is cheap
- **Hybrid achieves 52%** by skipping 70% of empty clusters, but cluster granularity is too coarse when data is extremely sparse

Medium Selectivity (5-20%). This is hybrid's sweet spot (rightmost cluster in Figure 5 and Figure 6):

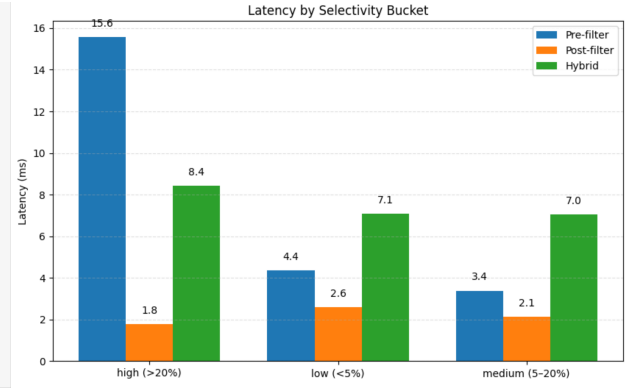

Figure 6: Latency by Selectivity bucket

- **Hybrid achieves 78% recall** at 7.0ms, dramatically outperforming post-filter's 20% while matching pre-filter's latency
- Predicate pushdown skips 100 empty clusters, performing only 2,500 distance computations versus 10,000 for post-filter
- Pre-filter maintains perfect recall at 3.4ms as 1,000 tracks remain manageable

High Selectivity (>20%). All methods achieve perfect recall (leftmost cluster in Figure 5and Figure 6), but latency diverges:

- **Post-filter dominates** (1.8ms)—8.7× faster than hybrid because abundant valid data ensures comprehensive results
- **Hybrid loses advantage** (8.4ms) as nearly all clusters contain valid data, eliminating pruning benefits
- **Pre-filter degrades** (15.6ms, tallest bar in Figure 6) due to exhaustive search over thousands of tracks

4.2.4 Key Insights. When Hybrid Excels:

- Medium selectivity (5-20%): Figure 5 shows 78% recall versus post-filter's 20%
- Variable workloads: Maintains 52-100% recall across all buckets (Figure 5)
- Tunable tradeoffs: Figure 3's frontier allows positioning between 8-15ms based on requirements

When Hybrid Struggles:

- Very low selectivity: 52% recall (Figure 5) trails pre-filter's perfect 1.00
- High selectivity: 4.7× slower than post-filter (Figure 6) with identical recall
- Perfect recall needs: Figure 4 shows 100% recall requires nprobe150, negating latency benefits

4.2.5 Limitations . **Recall-Latency Tradeoff:** Figure 4 shows diminishing returns—the final 12% recall improvement (88%→100%) costs double the latency (+7ms). Initial gains are steep (76%→88% for +0ms) but plateau quickly.

Selectivity-Dependent Performance: Hybrid's 52-100% recall range (Figure 5) means inconsistent quality across queries. Pre-filter's flat 1.00 line provides predictability that hybrid cannot match.

Complexity: Achieving Figure 5 Figure 6 Figure 4 performance requires 4× more code (200 vs 50 lines) for cluster management and bitmap operations, plus 10% memory overhead (40KB for 10,000 tracks).

Tuning Required: The nprobe=20 selection (Figure 4) reflects our latency-first priorities, but different applications need different configurations. Organizations must profile their workloads to find optimal settings.

5 LESSONS LEARNED

Over the course of this project, our approach evolved substantially as we discovered implmentation mistakes and incorrect assumptions. Initially, we attempted to evaluate hybrid methods using an approximate ground truth generated from the IVF-Flat index itself. This led to inconsistent recall reports and incorrect baselines, because any ANN-based "ground truth" systematically misses true neighbors. We then switched to a full IndexFlatL2 exact index, which resolved correctness issues but surfaced a second challenge that our pre-filter baseline was not achieving recall = 1.0 as expected. We discovered this was happening due to not considering a varied range of queries and were just testing it on exmaple queries.

We also tried multiple ways to implement the predicate pushdown method and were facing higher latency because our earlier version attempted to run FAISS search inside every cluster. Later we tried pre-allocating FAISS objects but found that reusing them across clusters did not reduce latency due to re-allocation and memory copying overhead. We tried to overcome these challenges and finally came up with the methodology mentioned in the report.

Through this project, we realized that the correctness of hybrid vector search is extremely sensitive to how ground truth is defined and how structured filters interact with ANN retrieval. The most important takeaway is that even slight inconsistencies in how filters are applied across methods can completely invalidate recall measurements. A key insight is that predicate pushdown yields significant performance benefits when filter selectivity is low to moderate. Another surprising finding was that the baseline post-filter approach can fail substantially even on exact L2 search unless the candidate pool is sufficiently large. This highlighted the importance of pool sizing in hybrid pipelines.

On the systems side, we realized that FAISS can introduce significant overhead when temporary indexes are created or search structures are reconstructed inside inner loops. Straightforward hybrid traversal often performs worse than even the simplest baselines. Achieving acceptable performance required reducing per-cluster FAISS calls, cutting down Python loop overhead, and eliminating unnecessary data movement.

We did imporve our methodology in terms of recall and latency but there are improvements that can be made to strengthen the system. First, replacing IVF-Flat with more expressive FAISS indexes such as IVFPQ, HNSW, or IVF-HNSW-PQ could significantly reduce the cost of exploring candidate clusters, especially for queries with broad filters. Second, at the metadata layer, richer cluster-level summaries could be precomputed instead of relying only on bitmaps, like range envelopes or bloom filters.

AI USE STATEMENT

Please note we used AI tools (GPT-5, Gemini) to conduct research on our implementation plan and to gain deeper understanding of some of the steps involved. We also used AI to assist with formatting the architecture figures parts of this report in proper Overleaf LaTeX.

GITHUB LINK

Link to GitHub repo: https://github.com/yangshengz88/Hybrid_Vector_Search/tree/main

REFERENCES

[1] Vespa Engineering Blog. 2022. Query Time Constrained Approximate Nearest Neighbor Search. https://blog.vespa.ai/constrained-approximate-nearest-neighbor-search/.

[2] ApX Machine Learning. 2025. Advanced Filtering Strategies: Pre vs. Post Filtering. https://apxml.com/courses/advanced-vector-search-llms/chapter-2-optimizing-vector-search-performance/advanced-filtering-strategies/. Accessed: 2025-09-25.

[3] Vatsal Mavani. 2020. Spotify Dataset (1921–2020). https://www.kaggle.com/datasets/vatsalmavani/spotify-dataset/data. Accessed: 2025-09-25.

[4] Zhixin Zhao, Xiaoyang Wang, Dongjie He, Jinhui Yuan, and Bin Cui. 2023. A Comprehensive Study of Hybrid Vector Search: When and How to Combine Vector with Attribute Filters. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991. https://doi.org/10.14778/3598581.3598594