Efficient Hybrid Search for Music Discovery

CS6400-A Project Milestone (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 a rapid growth over the years and there is a lot of scope in building a efficient and accurate search system that can take care of both semantic aspects of the audio and structured constraints of metadata. Traditional approaches do exist for the performing hybrid search. These methods perform a vector similarity search followed by a structurred filtering. Such methods depend on simple pre-filtering or post-filtering pipeilnes which often suffer from scalability bottlenecks and wasted computation [2] . For this project we looked into some improvements that can be made to search systems to improve performance and speed [1, 4].

In this project we will evaluate a hybrid music search system that fuses vector similarity over Spotify's 19-dimensional audio feature vectors with precise metadata constraints in a single query process. We are currently looking into two methodologies to achieve the same: (1) Intersect-then-rank, where vector and metadata are computed independently then intersected before ranking, and (2) Predicate Pushdown, where metadata filter are done in the approximate nearest neighbor (ANN) search to reduce candidates during search.

In this milestone, we have included our progress and results so far as well as our changes and plan to complete the rest of the project to achieve an efficient hybrid search for music discovery. We have also included the prelimnary exploratory data analysis we did and our implementation of the baseline methods.

2 PROPOSED APPROACH

Our project aims to build a hybrid music search system that combines vector similarity search over Spotify's 19-dimensional audio feature vectors (e.g., danceability, energy, valence, tempo) with precise metadata constraints. Instead of relying on pre-filtering or post-filtering, our core strategy is to fuse vector embedding search with structured metadata filtering in a single, efficient query. This approach pushes down metadata constraints directly into the search process, avoiding the bottlenecks associated with naive pre- or post-filtering methods.

To achieve this, we are currently exploring two primary hybrid search models: an intersect-then-rank strategy and predicate pushdown approach, where metadata constraints are applied during candidate generation to reduce expensive approximate nearest neighbor (ANN) computations.

3 ADDRESSING PROPOSAL FEEDBACK

In this section, we breifly address the feedback we received in our project proposal before we move on to discussing our implementation.

3.1 Predicate Integration with FAISS Traversal

In our initial proposal, the integration of predicate checks with FAISS traversal was not fully specified.

We plan to implement predicate evaluation at cluster level for the IVF-Flat index.

- Each inverted list (cluster) will be pre-associated with bitmasks representing eligible items for different structured filters.
- During FAISS traversal, we will skip clusters that fail the predicate before distance computation, reducing the candidate pool size and unnecessary distance calls.

This motivates our hybrid design by integrating predicates directly into the search process to balance latency and recall.

3.2 Fixing Embedding Model Early

Following the feedback we plan to finalize single enbedding representation to ensure fast progress and reproducible results. For the milestone, we commited to using feature-based 10-dimensional embeddings derived directly form the spotify metadata. (danceability, energy, valence, etc.). These embeddings are Normalized to [0,1] range, Lightweight for efficient FAISS indexing, suffieciently descriptive for similarity evaluation within our dataset**.**

4 DATA PREPARATION AND FEATURE ENGINEERING

This section includes the dataset and preprocessing information along with the initial exploratory data analysis (EDA). For Embedding dataset, used the Spotify dataset from Kaggle [3], which contains 170,653 vectors across 19 dimensions.

4.1 EDA results

We began by analyzing the Spotify Kaggle dataset (approx. 170K tracks) to understand data quality and feature distributions. As shown in Table 4, the dataset contains both numeric acoustic descriptors (energy, danceability, valence) and metadata fields (artists, year). We checked for missing and invalid values within out data and explored distributions and correlations within the data.

- **Feature Distribution:** Please refer to the feature distribution plots within the Figure 1. We were able to gain some important interpretations as mentioned in Table 2.
- **Outlier Detection:** Based on the results from the feature disctribution, we performed outlier detection for **Tempo, Duration and Popularity**. We found that Tempo had values greater than 200 and Duration had values greater than 16 mins as seen in Figure 2. Such outliers can dominate distance computations once features are normalized, skewing similarity in embedding or metadata space. For this reason

we applied Tempo and Duration filters before normalizing numeric columns within our data.

- **Correlation Heatmap:** The correlation heatmap Figure 3 reveals key relationships among the features. Energy and acousticness show strong negative correlation (-0.75), indicating that highly energetic tracks are rarely acoustic. Danceability and valence are moderately correlated (0.56), suggesting upbeat and "happy" songs tend to be more danceable. Meanwhile, popularity correlates positively with energy (0.49) but negatively with acousticness (−0.57), implying that high-energy, less acoustic tracks are generally more popular. These correlations help us understand feature redundancy and guide embedding design. For instance, highly correlated features might not add unique information during similarity search.

Overall, the EDA phase confirmed that the dataset is clean, diverse, and suitable for downstream modeling. After removing extreme outliers and understanding key feature relationships, we ensured that the remaining attributes are well-scaled and representative. These insights directly inform our next steps, normalizing features and generating compact numerical embeddings for efficient similarity search.

4.2 Feature Scaling and Embedding Generation

After data cleaning and outlier removal, we standardized all numeric features and encoded categorical variables to prepare the dataset for embedding generation and FAISS indexing. The goal was to ensure that all features contribute proportionally to similarity computations.

- **Numeric Feature Scaling:** Continuous attributes such as `danceability`, `energy`, `valence`, `tempo`, `acousticness`, `instrumentalness`, `liveness`, `speechiness`, `loudness`, and `popularity` were scaled to the range [0, 1] using Min–Max normalization. This ensures uniform contribution across dimensions and prevents high-magnitude features (e.g., `tempo`) from dominating distance metrics.
- **Categorical Feature Encoding:** Discrete fields such as `key`, `mode`, and `explicit` were label-encoded into integer representations. Non-numeric identifiers like `name`, `artists`, and `id` were preserved for interpretability and mapping during retrieval.
- **Embedding Construction:** We directly constructed embeddings from the normalized numeric features, forming the primary input to FAISS for similarity search.
- **Validation Checks:** Verified that all numeric features lie strictly within [0, 1], no missing or infinite values remain, and categorical fields match expected discrete ranges.
- **Output Artifacts:** The cleaned dataset was saved as "spotify_clean.parquet", while the embedding arrays were stored as "spotify_vectors_10d.npy". These became the foundation for FAISS indexing and hybrid retrieval evaluation as well as our baseline tests.

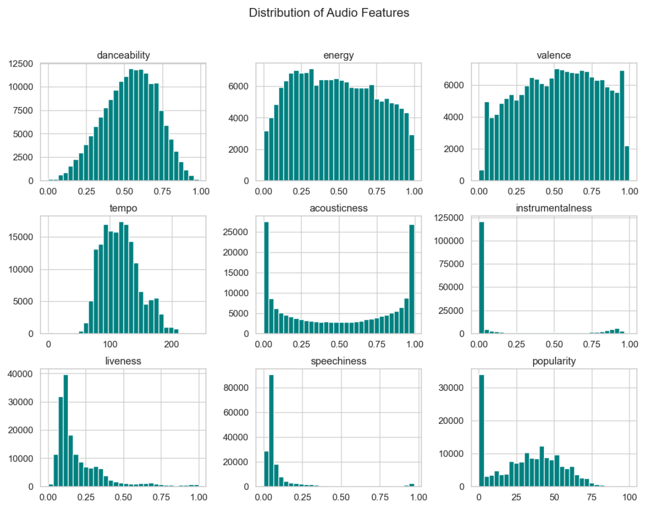


Figure 1: Distribution of Audio Features

Table 1: Metadata schema table.

Column	Range	Unique values	Data type
id	N/A	170653	string
valence	[0,1]	1733	float
year	[1921,2020]	100	integer
acousticness	[0,1]	4689	float
artists	N/A	34088	list of string
danceability	[0,1]	1240	float
duration(ms)	[5108,5400000]	51755	integer
energy	[0,1]	2332	float
explicit	[0,1]	2	integer
instrumentalness	[0,1]	5401	float
key	[0,11]	12	integer
liveness	[0,1]	1740	float
loudness	[-60, 3.85]	25410	float
mode	[0,1]	2	integer
name	N/A	133638	string
popularity	[0,100]	100	integer
release date	N/A	11244	string
speechiness	[0,0.97]	1626	float
tempo	[0,244]	84694	float

5 BASELINE SEARCH AND FILTERING BENCHMARKS

To evaluate similarity retrieval performance before implementing hybrid filtering, we established two baseline pipelines using the FAISS library on our normalized 10-dimensional track embeddings. Both pipelines retrieve the most similar tracks to a given query vector while incorporating structured metadata filters such as `year` and `explicit`.

- **Setup:** The FAISS index was built using `IndexFlatL2` for exact Euclidean distance search. Each track vector was mapped to its corresponding metadata (`id`, `name`, `artists`,

Table 2: Interpretation of the Feature Distributions in Figure 1.

Feature	Observation	Interpretation
Danceability	Roughly normal, centered around 0.5	Balanced, no scaling issues.
Energy	Slightly right-skewed, most tracks around 0.4-0.8	The range is normal
Valence	Fairly uniform distribution	Good diversity, helps for embedding-based similarity.
Tempo	Concentrated between 80-150 BPM	Typical of popular music but we further verify no negative or extreme values.
Acousticness	U-shaped, many highly acoustic and highly electronic tracks	Shows strong contrast between acoustic vs. studio-produced tracks.
Instrumentalness	Extremely right-skewed, majority near 0	Shows that most songs have vocals (low instrumentalness). Log scale when plotting can be used.
Liveness	Heavily right-skewed; most tracks <0.3	Shows most recordings are studio-based.
Speechiness	Sharp spike near 0, then smaller clusters near 0.3-0.4	Indicates few spoken-word or rap-style tracks.
Popularity	Skewed left, most songs have low popularity scores	Suggests long-tail distribution, only few tracks are highly popular.

`year`, `explicit`). For each experiment, we randomly selected multiple query tracks and applied both narrow (e.g., `year: 2000–2010`) and broad (e.g., `explicit: 0`) filters.

- **Pre-filtering Approach:** Metadata constraints were applied before FAISS search. The candidate pool was filtered using pandas, and similarity was computed only within this subset. This approach helps with lower computational cost but risks missing relevant items if filters are too restrictive. Typical latency observed: ~15–40 ms.
- **Post-filtering Approach:** FAISS search was run across the entire dataset, and metadata filters were applied on the retrieved top-K results. This ensures completeness but increases runtime as irrelevant candidates are also compared. Typical latency observed: ~2–4 ms for filtering after retrieval.
- **Observations:** For certain queries (`year: (2000–2010)`, `explicit:0`), both methods returned valid recommendations, but post-filtering often yielded very few (or no) matches for narrow constraints. This confirms that post-filtering alone caps achievable recall due to limited surviving candidates.
- **Insights and Next Steps:** The experiments helped us understand our data pipeline and established baseline latency and recall behavior. In the next stage, we plan to implement hybrid filtering techniques integrated with FAISS traversal, allowing metadata conditions to be checked dynamically during search.

Based on the above mentioned implmentation, we got some initial plots to understand how both the methods are working. Figure 4 shows that post-filter method achieves near-perfect recall with significantly lower latency compared to pre-filter approach. Such results are expected since filtering after retrieval allows FAISS to search the full vector space efficiently before applying conditions, whereas pre-filtering restricts the search space and increases lookup time.

In Figure 5, we also see Recall@K remains constant at 1.0 for all neighborhood sizes (K = 1–50), indicating that the IndexFlatL2 baseline performs exact nearest neighbor retrieval. This outcome is expected, as IndexFlatL2 computes full pairwise distances-thus serving as an upper-bound benchmark for future approximate and hybrid methods.

Table 3: Performance with different nprobe values

nprobe	Latency (ms)	Recall
1	0.254	1.000
5	0.090	1.000
10	0.067	1.000
20	0.084	1.000
50	0.187	1.000
100	0.499	1.000
200	0.807	1.000

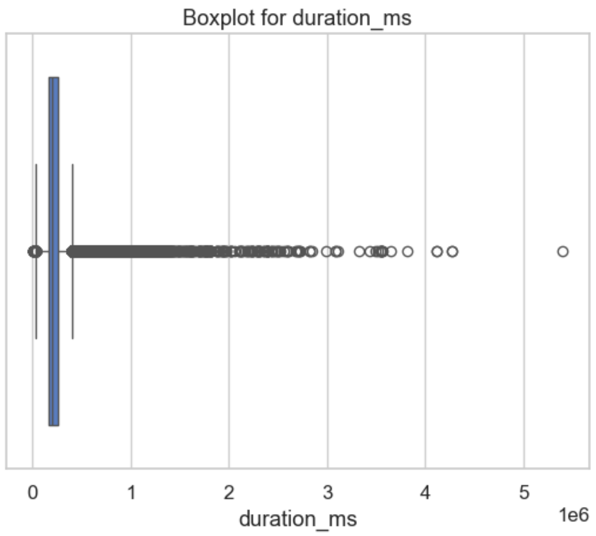
6 FAISS INDEX AND METADATA INTEGRATION

6.1 FAISS Index Construction

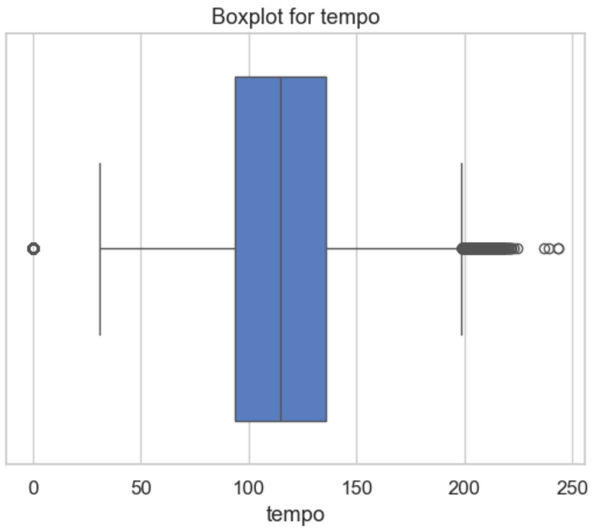
Our implementation focuses on building efficient FAISS indexes and complementary metadata structures to support hybrid search. We construct both a brute-force index (IndexFlatL2) for ground-truth recall evaluation and an IVF-Flat index for scalable approximate nearest neighbor (ANN) retrieval. The system automatically determines an optimal nlist parameter based on dataset size to ensure balanced clustering. To analyze performance trade-offs, we systematically test multiple nprobe values (1, 5, 10, 50, 100, 200), demonstrating how recall improves at the cost of increased latency. An example of searching for the top-10 nearest neighbors using different *nprobe* values based on a single sample track is shown in Table 3 . All indexes and their configurations are saved for reproducibility and later use.

6.2 Metadata Indexing

For metadata, we create efficient indexing strategies for both categorical attributes (e.g., artist, track name) and numeric attributes (e.g., year, tempo, popularity). Numeric range queries are supported

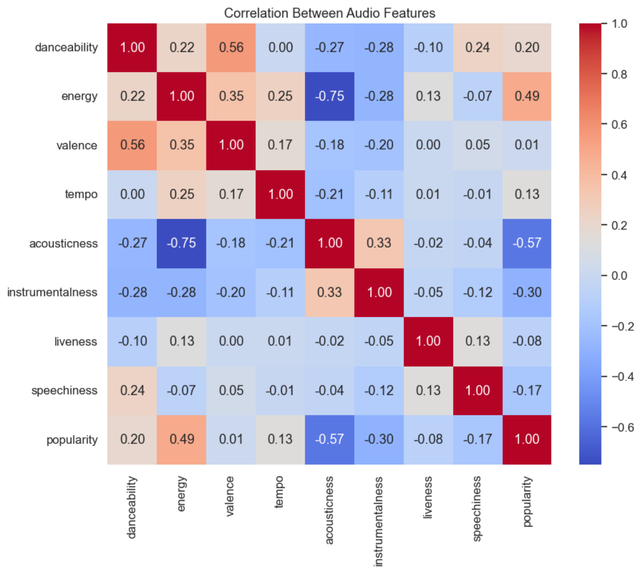


(a) Boxplot for track duration (in milliseconds).



(b) Boxplot for track tempo (in BPM).

Figure 2: Boxplots showing the distribution and outliers for (a) track duration and (b) tempo. Extreme outliers were filtered to ensure representative distributions.



	danceability	energy	valence	tempo	acousticness	instrumentalness	liveness	speechiness	popularity
danceability	1.00	0.22	0.56	0.00	-0.27	-0.28	-0.10	0.24	0.20
energy	0.22	1.00	0.35	0.25	-0.75	-0.28	0.13	-0.07	0.49
valence	0.56	0.35	1.00	0.17	-0.18	-0.20	0.00	0.05	0.01
tempo	0.00	0.25	0.17	1.00	-0.21	-0.11	0.01	-0.01	0.13
acousticness	-0.27	-0.75	-0.18	-0.21	1.00	0.33	-0.02	-0.04	-0.57
instrumentalness	-0.28	-0.28	-0.20	-0.11	0.33	1.00	-0.05	-0.12	-0.30
liveness	-0.10	0.13	0.00	0.01	-0.02	-0.05	1.00	0.13	-0.08
speechiness	0.24	-0.07	0.05	-0.01	-0.04	-0.12	0.13	1.00	-0.17
popularity	0.20	0.49	0.01	0.13	-0.57	-0.30	-0.08	-0.17	1.00

Figure 3: Correlation between features

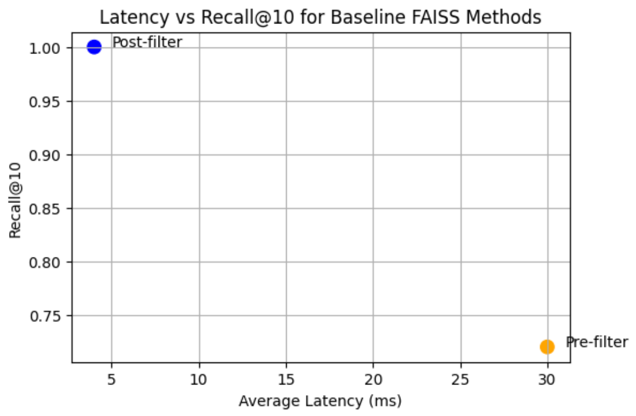


Figure 4: Latency vs Recall@10 for baseline FAISS Methods

through sorted arrays combined with binary search for fast lookups. To enable high-speed filtering during query execution, we implement a MetadataFilter helper class that uses bitmap-based filtering, allowing O(1) checks for candidate validity. Additionally, utilities are provided to compute filter selectivity, which helps in evaluating query efficiency and optimizing predicate pushdown strategies.

7 HYBRID METHOD — PREDICATE PUSHDOWN

The core contribution of our system is the implementation of predicate pushdow**n** during FAISS traversal, enabling an efficient hybrid search that combines vector similarity with metadata constraints. To achieve this, we precompute cluster assignments for all embeddings and build cluster-level filter bitmaps, allowing rapid determination of whether a cluster contains any valid candidates before probing. This optimization significantly reduces unnecessary distance computations by skipping entire clusters that do not satisfy the filter criteria.

This functionality is encapsulated in the *HybridSearcher* class, which supports multiple search strategies for comprehensive evaluation:

- **Predicate Pushdown**
- **Post-Filter Baseline**
- **Pre-Filter Baseline**

To ensure transparency and enable performance analysis, the system tracks detailed statistics, including the number of clusters probed and skipped, distance computations performed, and query

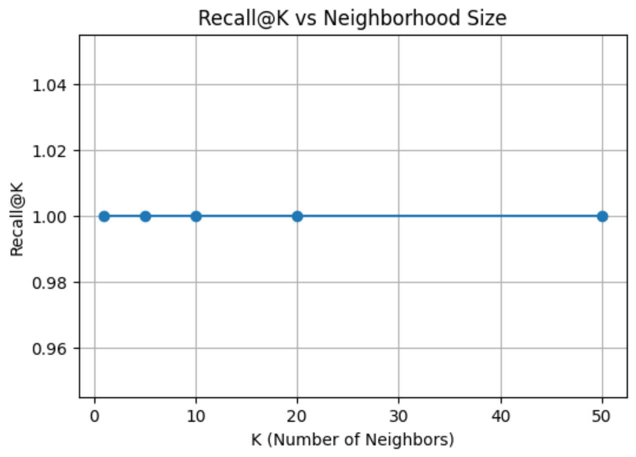


Figure 5: Recall@K vs Neighborhood size

Table 4: Comparison of predicate-pushdown (hybrid) and baseline FAISS methods with cluster level filtering

Stats	hybrid	Post-Filter	Pre-Filter
latency	53.70ms	32.41ms	27.63ms
distance computations	64	100	13,940

latency. These metrics provide insight into the trade-offs between accuracy, efficiency, and memory overhead, validating the effectiveness of predicate pushdown compared to baseline approaches. For evaluation, we tested a single query track with nprobe = 20, and the result is summarized in Table 4. The results did not meet our expectations, so we plan to conduct additional testing and may revise our hybrid method. For more details, please refer to the Risk Assessment and Mitigation section.

8 QUERY WORKLOAD

Below are example queries designed to compare the performance of the Intersect-then-Rank and Predicate Pushdown strategies under different conditions.

High-selectivity queries (narrow filters): These queries have precise metadata constraints that match only a small number of tracks. They test the efficiency of filtering early, a key advantage of the Predicate Pushdown method.

(1) **Point query with multiple exact matches**
- **Audio feature constraint:** Search for tracks acoustically similar to a specific query song.
- **Metadata constraint:** Filter for songs by a specific niche artist and from a specific release year
- **Query example:** Find songs similar to [Track-X] by artist "Niche Artist" released in year 2022.

Low-selectivity queries (broad filters): These queries have less restrictive metadata filters that match a large portion of the dataset. They test how efficiently the system can perform the ANN search without being bogged down by a slow, large-scale filter, which challenges simpler baseline methods.

(1) **Range query with a broad range**
- **Audio feature constraint:** Search for tracks similar to a specific query song.
- **Metadata constraint:** Filter for songs with a very broad tempo range.
- **Query example:** Find songs similar to [Track-Y] with a tempo between 80 and 150 bpm.

(2) **Multi-column query with broad filters**
- **Audio feature constraint:** Search for tracks similar to a specific query song.
- **Metadata constraints:** Filter for songs from a popularity higher than some threshold and released after a specific year.
- **Query example:** Find songs similar to [Track-Z] with popularity higher than 50, released after 2010.

Combined selectivity queries These queries blend specific and general filters to stress-test the system's performance. The outcome will depend on which filter the execution strategy prioritizes.

(1) **Mix of narrow and broad filters**
- **Audio feature constraint:** Search for tracks similar to a specific query song.
- **Metadata constraints:** Filter for a specific record label (narrow filter) and a broad range of song duration (broad filter).
- **Query example:** Find songs similar to [Track-A] by artist "Niche Artist" with a duration between 3 and 5 minutes.

(2) **Multi-column range queries**
- **Audio feature constraint:** Search for tracks similar to a specific query song.
- **Metadata constraints:** Filter for a specific danceability range and a specific energy range.
- **Query example:** Find songs similar to [Track-B] with danceability between 0.6 and 0.8 and energy between 0.7 and 0.9.

9 EXPECTED OUTCOMES

9.1 Expected Results and Analysis

We anticipate the following performance patterns based on each method's design characteristics:

- **Post-filtering** is expected to achieve the lowest latency, as it performs a fast, unfiltered vector search. However, we predict its recall will be severely impacted for high-selectivity queries, as the true nearest neighbors may fall outside the initial top-K candidates retrieved before filtering.
text
- **Pre-filtering** should show variable performance. For low-selectivity queries, it may maintain good recall but with higher latency than post-filtering. For high-selectivity queries, its latency should improve (due to smaller search space) but recall will suffer significantly if the pre-filtered set excludes true nearest neighbors.
- **Predicate Pushdown (Ours)** is designed to fundamentally outperform both naive baselines by integrating metadata constraints directly into the search process. We expect it to

achieve superior recall compared to post-filtering by ensuring that relevant clusters are prioritized during traversal, while maintaning better latency than pre-filtering by eliminating unnecessary distance computations. This advantage should be most evident in high- and combined-selectivity scenarios.

9.2 Validation Plan

To verify these expectations, we will:

(1) Implement the full query workload
(2) Execute each query across all three methods
(3) Measure latency and recall metrics systematically
(4) Perform statistical significance testing on the results
(5) Analyze the scalability of each method as we increase dataset size

We believe that validating these expected results will demonstrate a clear advantage for our hybrid method, moving us closer to a search system that is both intelligent and efficient enough for real-world use.

10 EVOLVING METHODOLOGY

The initial phase of our project aims toward establishing strong pipeline for baslines through pre- and post-filtering using the IndexFlatL2 index. Here, we confrimed that post-filtering achieves low latency (2-4ms) and perfect recall (approx. 1.0), validating out FAISS setup and embedding quality. Post-filtering retrieved only the top-k neighbors before applying filters, which is acceptable for exact search but suboptimal for approximate methods. Moving forward, we will retrieve a larger candidate pool (via nprobe or ef_search) before filtering to preserve recall under hybrid and approximate search settings. We even implemented the pipeline for predicate pushdown and recorded our observations for the same with IVF-Flat method.

In parallel, we want to **implement an IVF-PQ index** to support approximate retrieval and tune parameters such as nlist and nprobe. Pre-indexed Boolean masks for categorical filters (e.g., explicit flag, mode) and sorted ranges for numeric filters (e.g., year, tempo) will be integrated into both hybrid methods.

To further enrich the vector space, we plan to extend embeddings beyond purely numeric features. Future iterations will incorporate **artist, track name**, and other textual metadata using transofrmer-based or averaged word embeddings. We believe this will allow the hybrid search to capture both acoustinc and semantic similarity across tracks.

Lastly, for consistent and fair comparisons across all the methods, we are also working on **structured query workload** combining both narrow and broad filters. Each query will include metadata constraints alongside vector similarity search to represent realistic music retrieval tasks.

11 RISK ASSESMENT AND MITIGATION

Our baseline evaluation showed that post-filtering acheived exceptionally low latency while maintaining perfect recall. Although this can be expected while using `IndexFlatL2`, it reduced the expected performance gap for hybrid methods. As a result, our we want to focus more on marrow filters and approximate indexes as discussed above.

One potential risk to the success of our project is that the chosen ANN indexing method, IVF-Flat, may not deliver a significant performance improvement over baseline approaches on the full dataset. This could limit the effectiveness of our hybrid search strategy and impact overall evaluation metrics. To mitigate this risk, we plan to experiment with IVF-PQ (Product Quantization), which offers better memory efficiency and faster search by compressing vectors while maintaining reasonable accuracy. If IVF-PQ does not yield the desired improvement, our backup plan is to adopt HNSW (Hierarchical Navigable Small World) indexing, a graph-based ANN method known for high recall and low latency. This tiered approach ensures that we have alternative strategies to maintain performance goals even if the initial method underperforms.

Integrating structured filtering logic directly into the FAISS framework introduces additional implementation complexity and potential runtime overhead. Since FAISS is optimized for pure vector operations, adding metadata-based conditions may slow traversal or distance computations if not carefully handled. To mitigate this, we plan to start with a modular Intersect-then-Rank implementation before embedding filters within FAISS. Each stage will be benchmarked independently to identify bottlenecks and ensure that hybrid integration improves overall efficiency rather than reducing performance.

Overall, while early results validate system correctness and speed, the next phase focuses on stress-testing hybrid methods across approximate search configurations and diverse workloads to understand practical efficiency acheived.

12 TEAM UPDATE AND TIMELINE ADJUSTMENT

Our team initially consisted of four members, however, one member had to withdraw from the course due to personal reasons. This change has slightly redistributed the workload among the remaining three of us, leading to minor adjustments in our internal task timelines. While we were hoping to have completed testing baseline and hybrid with consistent finalised query workloads, we are still working on the same and aim to include it in our final results. Despite this change, we have maintained steady progress and coordinated responsibilities effectively so far. We anticipate completing all remaining implementation and analysis components on schedule for the final report submission. We also plan to create a github repository with the final python scripts to make this porject reproducible.

13 AI USE STATEMENT

Please note we used AI tools 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 parts of this report in proper Overleaf LaTeX.

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