ECG similarity search engine

Efficient and flexible retrieval of ECGs based on model-derived metrics

Hèlia Brull Corretger

22/06/2025

1. Brief summary **3**
2. System architecture **3**
2.1 Data generation 3
2.1.1 Risk-of-condition scores 3
2.1.2 Beat-type proportions 4
2.1.3 CNN or transformer-derived ECG embeddings 4
2.1.4 Heart rate 4
2.2 Preprocessing 4
Figure 1: 2-dimensional PCA of the pre-processed data 6
2.3 Indexing 6
2.3.1 Overview of index types 7
2.3.2 Single indexing strategy 7
2.3.2 Hybrid indexing strategy 8
2.4 Querying 8
2.4.1 Query inputs 9
2.4.2 Query strategy selection 9
2.4.3 Single indexer query execution 9
2.4.4 Hybrid indexer query execution 9
3. Results and scalability analysis **10**
3.1 Benchmarking setup 10
3.2 Query time across parameters 11
3.3 Individual trends 11
3.3.1 Query time vs dataset size (N) 12
3.3.2 Query time vs number of feature groups (n_groups) 13
3.3.3 Query time vs top-k retrieved (top_k) 14
3.4 Summary of findings 14
4. Simplifications and trade-offs **15**
4.1 Simplifications 15
4.1 Trade-offs 15
5. Future work and production considerations **16**
5.1 Future work 16
5.2 Production considerations 16
6. Codebase and reproducibility **17**
7. References **17**

1. Brief summary

This project presents a scalable ECG similarity search engine designed to retrieve ECGs similar to a given query based on user-defined combinations of ML-derived metrics. The system simulates structured synthetic data and supports two indexing strategies using FAISS: a **Single Indexer** for full-vector queries and a **Hybrid Indexer** for group-specific, weighted similarity.

Users can flexibly select feature groups (e.g., heart rate, embeddings) and retrieve top-k similar ECGs with sub-second query times, even at large scale (up to 1 million ECGs). The system is modular, interpretable, and benchmarked across dataset sizes, query parameters, and feature combinations.

While designed as a proof-of-concept, it establishes a solid foundation for clinical deployment, with potential extensions including adaptive weighting, richer features, and distributed indexing.

2. System architecture

The architecture of the ECG similarity system is modular and designed to support scalable and clinically interpretable similarity search. It comprises four main stages: (1) data generation, (2) feature preprocessing, (3) FAISS-based indexing, and (4) querying.

2.1 Data generation

The system begins by generating synthetic data for ECGs, including unique identifiers and simulated machine learning model outputs. Rather than relying on unstructured noise, I defined **six clinically grounded clusters**, each corresponding to a cardiovascular condition: **normal, AFib-prone, bradycardia, tachycardia, ischemia**, and **PVC-heavy**. These clusters were designed to reflect real-world ECG heterogeneity and align with prior research, particularly work conducted by Idoven.

Following a brief literature review, I selected a diverse set of clinically relevant and interpretable features commonly found in ECG-based machine learning systems: risk-of-condition scores [1], beat type classification (AAMI EC57 standard) [2], CNN or Transformer-derived ECG embeddings [3], and interval features such as heart rate.

Let us briefly describe each feature in the following subsections.

2.1.1 Risk-of-condition scores

Each ECG receives probabilistic scores from five hypothetical binary classifiers targeting **AFib, bradycardia, tachycardia, PVCs, and ischemia**. These scores are sampled from **Beta distributions**, which flexibly control variance while preserving the [0, 1] bounds of probability outputs. This choice is further supported by literature showing that Beta distributions are well-suited for simulating calibrated classifier outputs and modeling uncertainty beyond standard sigmoid transformations [4].

To reflect disease prevalence and model uncertainty:

- A score of 0.85 is assigned to the relevant cluster for each condition.
- 0.05 is assigned to all others (including "normal").
- A minor overlap is introduced, for the sake of simulating real-world comorbidity: e.g., PVC-heavy has 0.3 ischemia risk, and ischemia has 0.3 PVC risk, inspired by [5].

2.1.2 Beat-type proportions

We simulate the morphological composition of ECGs using beat type proportions from the **AAMI EC57** beat type taxonomy: **N** (normal), **S** (supraventricular), **V** (ventricular ectopic), **F** (fusion), **Q** (unknown).
Each ECG's composition is drawn from a **Dirichlet distribution** with cluster-specific concentration parameters (e.g., high **V-beats** in PVC-heavy [6], more **Q/S** beats in AFib-prone [7]).

Note that beat type annotations take place at the beat level, and that 10-second ECGs contain few beats (~6–20 depending on the heart rate), making fractional proportions somewhat unrealistic. A more faithful simulation would use beat count (from heart rate), sample with a multinomial distribution, and normalize. For simplicity, we use continuous proportions here.

2.1.3 CNN or transformer-derived ECG embeddings

Each ECG is also associated with a **dense, low-dimensional embedding** (64-dimensional by default) that simulates the **latent feature space** typically produced by convolutional neural networks or transformers trained on raw waveform input. For each cluster, I define a **random embedding centroid** and draw ECG-specific embeddings from a **multivariate Gaussian** centered at that mean, scaled by **cluster-specific variance** to reflect intra-class variability.

2.1.4 Heart rate

Lastly, each ECG includes a scalar heart rate value, modelled as a normally distributed variable with a **cluster-specific mean and variance**, and clipped to remain within physiologically plausible bounds (e.g., between a minimum and maximum heart rate). We assigned an **elevated heart rate in tachycardia or AFib-prone clusters**, and lower rates in the **bradycardia cluster.**

Although heart rate may not always be the direct output of a machine learning model, it represents a fundamental **interval-based feature** frequently used in ECG analysis and heartbeat classification. Including it in the feature set ensures the system can incorporate both morphological and temporal information.

2.2 Preprocessing

The preprocessing stage prepares heterogeneous ECG-derived features for indexing and similarity search. This is crucial for ensuring that features of varying types and scales (e.g., heart rate vs. CNN embeddings) can be meaningfully compared in a similarity search engine.

The preprocessing pipeline handles the four distinct feature groups:

1. Heart rate (scalar, continuous)

Heart rate is standardized using a standard scaler to ensure zero mean and unit variance. This avoids disproportionate influence during distance computation, especially since heart rate is on a different scale than embeddings or probabilities.

2. Risk scores (multivariate, probabilistic)

Each condition-specific risk score is individually standardized using its own standard scaler. This preserves independence among scores and aligns with how clinical risk models might output uncorrelated probability estimates. The standardized scores are concatenated into a single vector.

3. CNN-derived embeddings (high-dimensional)

CNN embeddings (typically ~64 dimensions) are first standardized across the dataset, then reduced to a lower-dimensional space using **Principal Component Analysis (PCA)**. This enhances computational efficiency and captures the most informative axes of variation (e.g., we reduce to 5 components). The transformed embedding is then appended to the final vector.

4. Beat-type proportions (multivariate, already normalized)

Beat-type proportions are already constrained to [0, 1] and sum to 1 (via Dirichlet sampling). These are used as-is, without additional transformation, under the assumption that their distribution is already harmonized.

The 2D PCA plot of preprocessed features in Figure 1 reveals clear separation between most ECG clusters, validating that the synthetic data generation and preprocessing steps preserved discriminative structure across simulated clusters. Some overlap is observed between ischemia and PVC-heavy clusters (expected due to shared simulated features) and between ischemia and bradycardia, which may warrant further investigation.

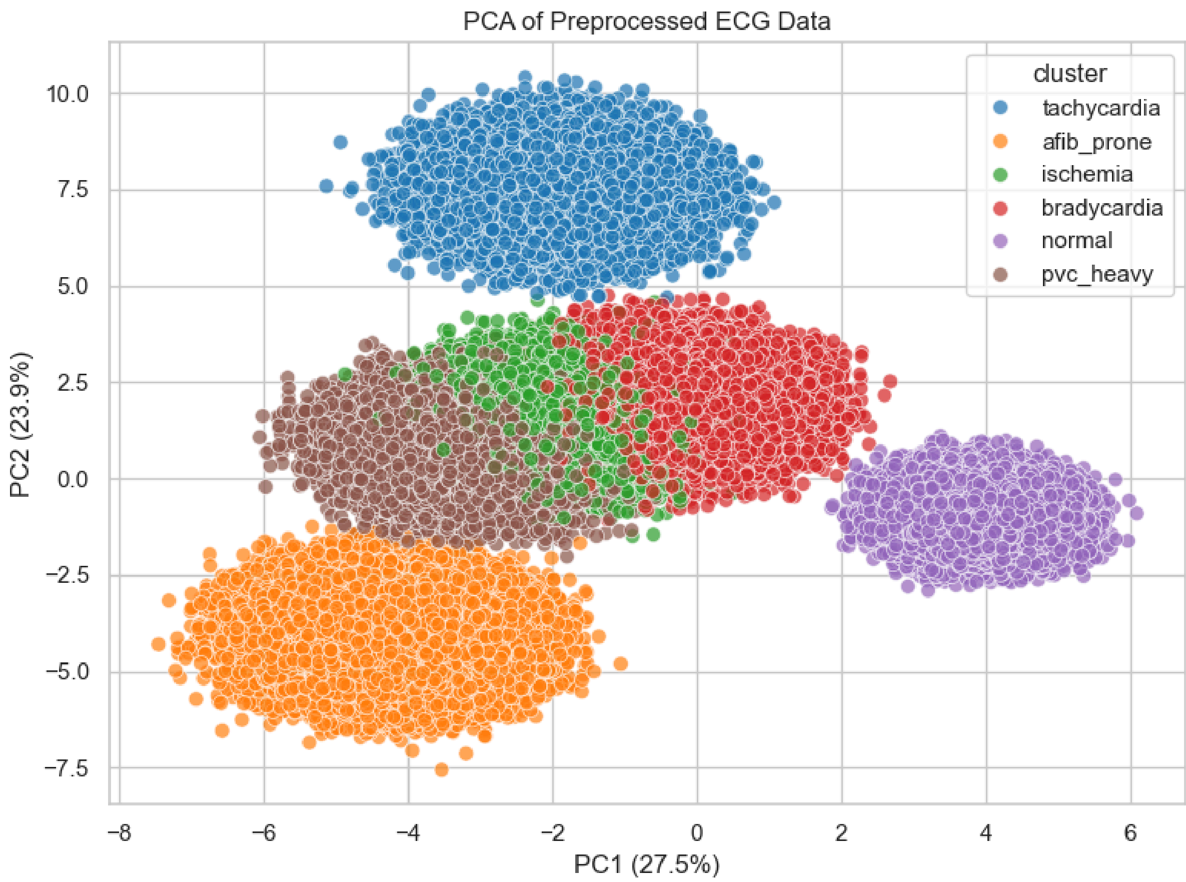

Figure 1: 2-dimensional PCA of the pre-processed data

2.3 Indexing

To enable fast and scalable similarity queries over ECG feature vectors, I use **FAISS** (Facebook AI Similarity Search) [8], a high-performance library for efficient nearest-neighbour search in high-dimensional spaces. FAISS supports both exact and approximate search, is optimized for CPU/GPU execution, and has been successfully applied in ECG retrieval tasks [9].

A practical tutorial by Pinecone [10] was particularly useful in shaping the system's indexing and retrieval setup.

The system implements two indexing strategies:

- A **Single Index** for full-feature similarity search
- A **Hybrid Index** for feature group–specific similarity search

These indexes are wrapped in the Python classes `SingleIndexer` and `HybridIndexer`, respectively.

A coordinating class, `SimilaritySearcher`, dynamically selects the appropriate strategy based on the query type.

2.3.1 Overview of index types

The indexing system supports two FAISS index types:

- `IndexFlatL2`: Performs **exact brute-force nearest neighbour search** using L2 (Euclidean) distance. It guarantees accurate results but does not scale well to large datasets due to its linear search time [11].
- `IndexHNSWFlat`: Implements **approximate nearest neighbour search** using a **Hierarchical Navigable Small-World (HNSW) graph** [12]. This structure enables sublinear query times [11] and is highly efficient for large-scale retrieval, at the cost of minor accuracy trade-offs. Performance of HNSW can be tuned via:
  - `M`: This parameter controls the **number of connections (edges)** each node in the graph has. A higher `M` increases **recall** and improves search quality but also makes the index larger and slower to build.
  - `efSearch`: This parameter determines the **number of candidate nodes** considered during query time. A larger `efSearch` value increases **search accuracy**, at the cost of **longer query time**.

Index selection is guided by the dimensionality and scale of each feature group:

- `IndexFlatL2` is preferred for low-dimensional vectors (e.g., scalar `heart_rate`), provided the dataset size and number of requested neighbours (`k`) remain modest.
- `IndexHNSWFlat` is used for higher-dimensional vectors (e.g., embeddings, risk scores, beat-type proportions) where approximate search is more efficient.

2.3.2 Single indexing strategy

The SingleIndexer class manages global similarity search using a single FAISS index built from the entire feature matrix. This method is well-suited to use cases where all features are treated as equally important, enabling straightforward, global comparisons across ECGs.

Index construction

- Supports both `IndexFlatL2` and `IndexHNSWFlat`
- Data is added in batches for scalability

Strengths

- Very fast for high-dimensional data with HNSW
- Simple and effective when all features are equally weighted
- Useful as a baseline or default retrieval strategy

Weaknesses

- Not interpretable when feature contributions vary
- Inefficient if only a subset of features is relevant to the query
- Less flexible in supporting task-specific or clinically grounded similarity definitions

2.3.2 Hybrid indexing strategy

The `HybridIndexer` class enables modular similarity search by independently indexing each **feature group** (e.g., `heart_rate`, `risk_scores`, `embedding`, `beat_props`). This allows selective querying over a subset of features and supports **interpretability**, **control**, and **flexibility** in similarity criteria.

Index construction

- Separate FAISS indexes are built per group
- Each group independently uses either `IndexFlatL2` or `IndexHNSWFlat`, based on dimensionality and the scale of the dataset
- Group-specific slices are stored to enable efficient querying
- Data is added in batches for scalability

Special handling: risk scores

All five `risk_*` scores are stored and indexed together as a unified `risk_scores` block. This reduces overhead, simplifies logic, and is justified by the structure of the synthetic dataset, where each ECG cluster is characterized by a high value in a single risk dimension. As a result, the combined `risk_scores` vector is both efficient to index and discriminative, capturing cluster-level clinical signals in a compact form.

Strengths

- Enables fine-grained, interpretable queries over selected feature groups
- Highly flexible: users can specify both selected groups and custom weights
- Scales well with large datasets, provided queries involve a limited number of groups

Weaknesses

- More complex query logic and implementation overhead
- Higher memory usage due to multiple indexes
- Performance depends on the quality and balance of per-group normalization and aggregation

2.4 Querying

The querying component retrieves ECGs most similar to a given input vector based on user-defined criteria. It supports flexible, task-specific retrieval by allowing dynamic selection of feature groups and customizable weighting.

2.4.1 Query inputs

The system accepts the following parameters at query time:

- `query_vec`: a single preprocessed ECG vector.

- `top_k`: number of most similar results to retrieve.
- `selected_groups`: list of feature groups to include in the similarity computation.
- `weights`: optional dictionary assigning a custom importance (weight) to each selected group.
- `hybrid_margin_factor`: optional multiplier that determines how many candidates each group returns during hybrid search before merging.

2.4.2 Query strategy selection

The `SimilaritySearcher` class routes each query to the appropriate indexing engine:

- If **all feature groups** are selected → use `SingleIndexer` for full-vector search
- If **a subset of groups** is selected → delegate to `HybridIndexer`

Additional logic:

- Individual `risk_*` inputs are mapped to `risk_scores`
- If multiple `risk_*` weights are given, the maximum is assigned to `risk_scores`

2.4.3 Single indexer query execution

When using the `SingleIndexer`, no group-specific handling is needed. The full preprocessed ECG vector is queried directly against the single FAISS index. The top-*k* most similar entries are returned based on L2 distance.

2.4.4 Hybrid indexer query execution

When the `HybridIndexer` is used, the query goes through the following steps:

1. Selection of groups

When a query is submitted, users may specify a subset of `selected_groups`, such as `["embedding", "heart_rate"]`. The system determines which groups are selected and maps `risk_*` to `risk_scores` if needed.

2. Group-wise FAISS search

- If only one group is selected, a direct top-k search is executed using that group's index
- Otherwise:
  - Each selected group's index is queried independently using its slice of the query vector
  - Each group returns `top_k × margin_factor` candidates
  - Distances are normalized (Z-score) per group to handle scale differences

3. Score aggregation

- Candidate lists are merged

- A weighted sum of normalized distances is computed for each candidate
- Missing results from a group are penalized with the group's max normalized score

4. Final ranking

- Candidates are sorted by total score (lower is better)
- Top-k most similar ECGs are returned

3. Results and scalability analysis

3.1 Benchmarking setup

To assess the scalability of the ECG similarity search system, we conducted a set of controlled experiments varying key parameters:

- `N`: total number of ECGs in the index (10K, 100K, 1M, 10M)
- `top_k`: number of results retrieved (100, 500, 1000)
- `n_groups`: number of feature groups selected in the query (from 1 to 4)

For each configuration, we randomly selected 5 query ECGs and measured the query time.. All experiments were conducted on a standard laptop CPU, and the indexing phase was excluded from timing measurements.

For further details, please check the Jupyter Notebook `ecg_similarity_search_benchmarking.ipynb`. Please note that this notebook is intended as a **secondary, exploratory analysis tool** to support the main findings and is not designed for standalone execution.

3.2 Query time across parameters

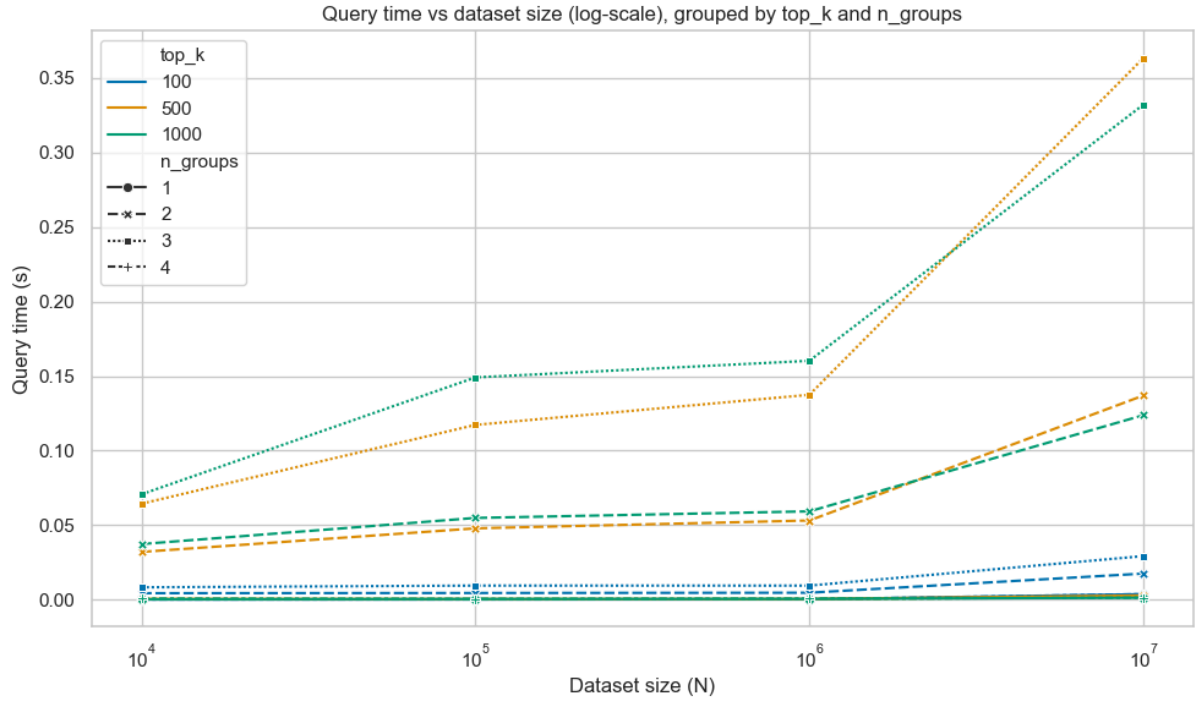

Figure 2: Query time across parameters

We jointly visualize the effect of N, top_k, and n_groups on query time. As expected:

- **Query time increases with** N due to the need to search over larger databases.
- **Query time increases** with top_k due to larger candidate retrieval and ranking.
- **Query time increases** with n_groups due to multiple index lookups and aggregation.

All configurations remain well under 1 second, satisfying the performance constraint.

3.3 Individual trends

We separately analyze average query time across the three axes.

3.3.1 Query time vs dataset size (N)

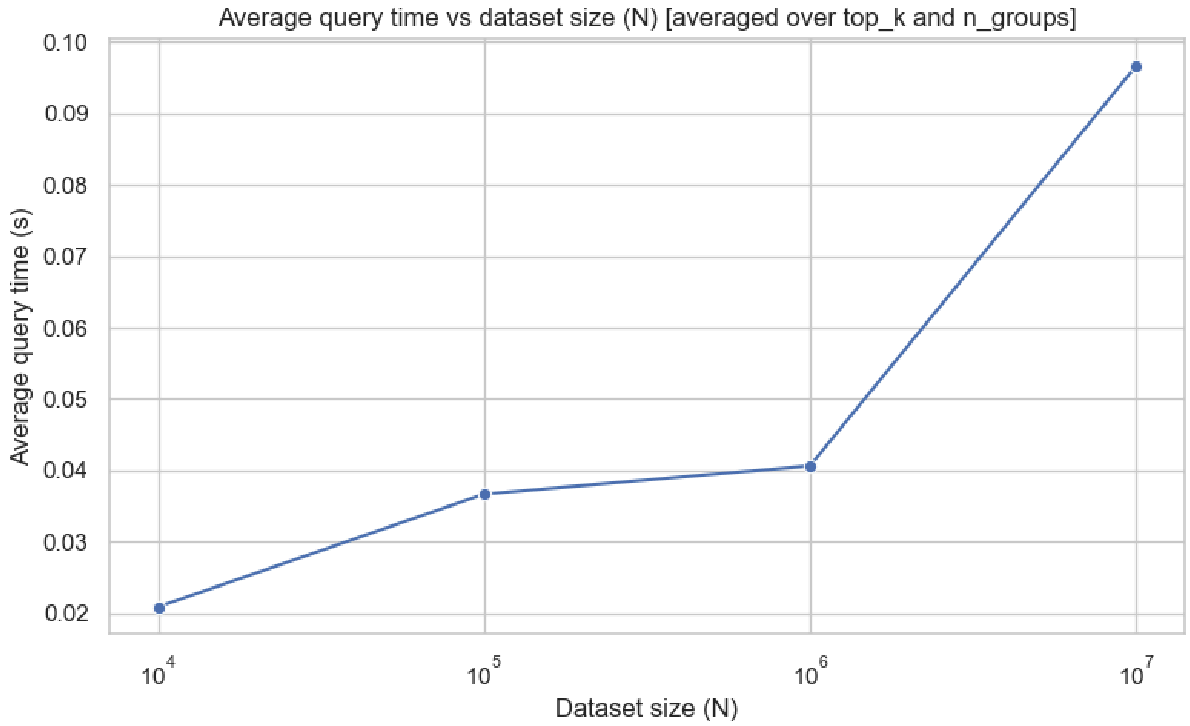

Figure 3: Average query time vs dataset size

Average query time increases with dataset size, remaining well below the 1-second constraint even at 10 million ECGs. Performance scales sub-linearly up to 1 million, with a steeper but still manageable increase at 10 million, demonstrating the system's scalability under realistic growth scenarios.

3.3.2 Query time vs number of feature groups (n_groups)

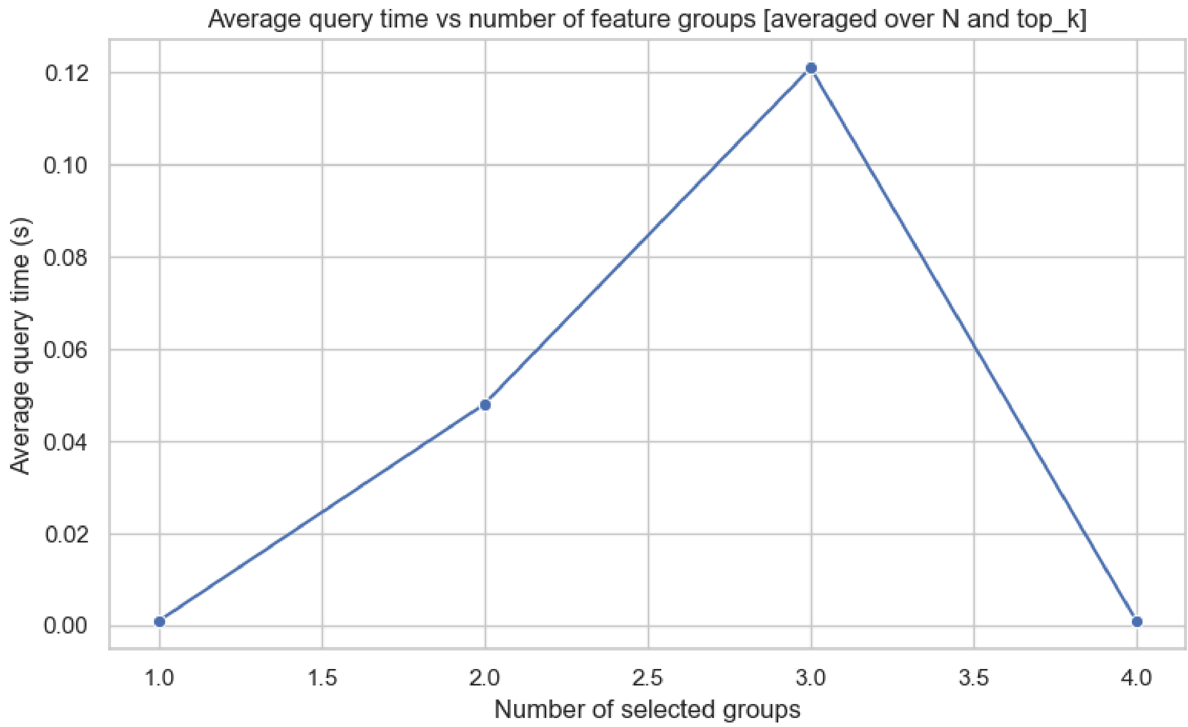

Figure 4: Average query time vs number of feature groups

Query time increases as more feature groups are selected due to the overhead of multiple index lookups, margin expansion, and score aggregation. However, when all groups are selected, the system switches to the `SingleIndexer`, performing a single search without margin expansion and reranking, leading to a substantial drop in query time.

3.3.3 Query time vs top-k retrieved (`top_k`)

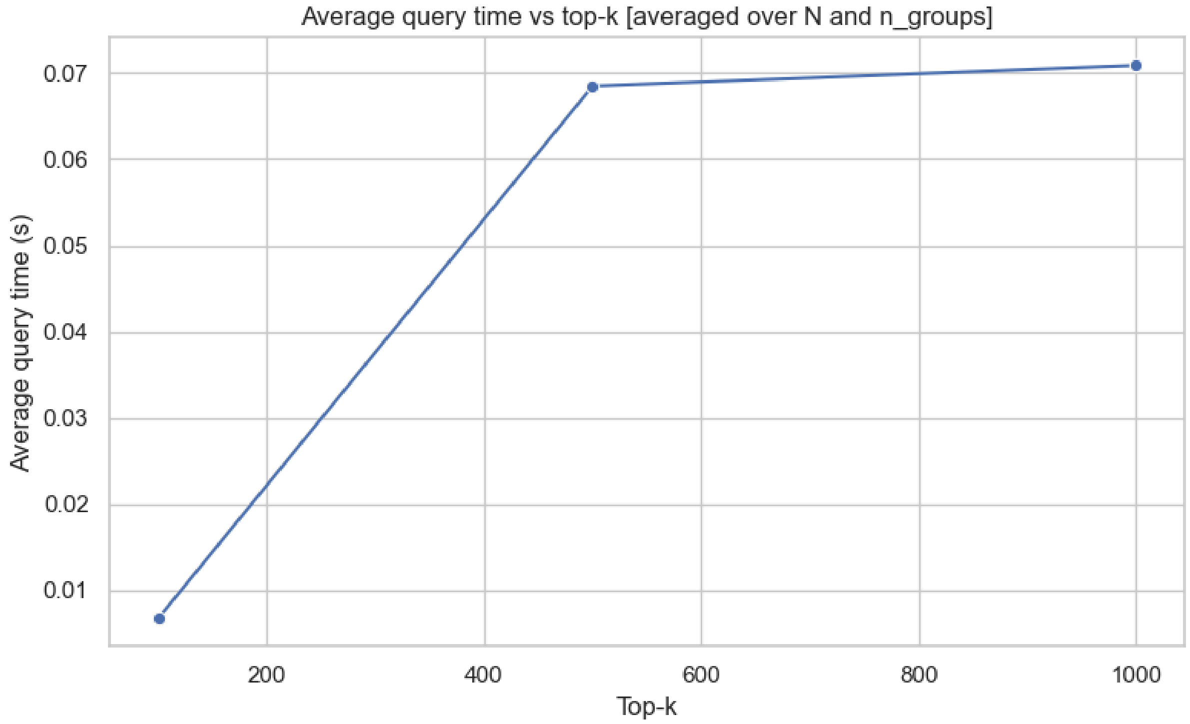

Figure 5: Average query time vs top-k

Query time grows with `top_k` as more candidates must be retrieved, scored, and sorted. However, the increase seems to be sublinear: the jump from 100 to 500 results in a significant time increase, while the increase from 500 to 1000 is marginal. This trend may reflect optimizations within the FAISS search process, but further profiling would be required to confirm that.

3.4 Summary of findings

- The system consistently achieves **query times under 1 second** across dataset sizes **up to 10 million ECGs**.
- **Query time increases** with the **number of feature groups** due to multiple index lookups and aggregation (search of `top_k*margin_factor` for each group).
- **Query time scales sublinearly** with both **dataset size** and the number of **requested neighbors** (`top_k`).

The hybrid design allows for flexibility, enabling users to trade off speed and granularity by adjusting parameters like `efSearch` and `M` for HNSW, as well as the `margin_factor` used in the `HybridIndexer`'s aggregation logic. While these were held constant in our experiments for the sake of simplicity, systematically exploring their effect on recall, latency, and resource usage would be a valuable direction for future work.

Overall, the system exhibits **strong scalability and performance**, indicating **readiness for deployment at even larger scales with minimal optimization**.

4. Simplifications and trade-offs

4.1 Simplifications

- **Synthetic data assumptions**: The ECG dataset is entirely synthetic, generated using fixed and arbitrary statistical parameters such as Gaussian-distributed heart rates and Beta-distributed risk scores. Clusters are cleanly separable in the synthetic space, which simplifies evaluation but does not reflect the overlap and ambiguity found in real-world clinical ECG data.

- **Beat-type proportions as continuous values:** Beat-type composition is represented using continuous proportions sampled from a Dirichlet distribution, which ignores the discrete and count-based nature of beat occurrences in short 10-second ECG recordings.

- **Assumption of model validity**: We assume that all simulated ML model outputs are valid, well-calibrated, and comparable across ECGs. In practice, the reliability of each model (depending on its accuracy, generalization, or calibration) would directly affect how much influence its corresponding metric should have in the similarity search.

- **Limited metric diversity**: Only four types of features are simulated: heart rate, risk scores, embedding vectors, and beat-type proportions. A production system would need to incorporate a broader set of clinical and morphological metrics, including interval-based features (e.g., PR, QT), waveform-level model outputs, etc.

- **Missing clinical context:** Patient-level metadata, such as patient ID, age, or prior clinical history, is not incorporated in this proof-of-concept. In real-world systems, such contextual information would be critical for personalizing similarity search and avoiding bias from repeated samples from the same individual.

- **Missing technical metadata:** The current implementation does not record technical metadata such as model version, feature extraction date, or code commit hash. In production, this metadata is essential for reproducibility and debugging.

- **No update mechanism:** The system assumes that all ECGs are indexed once and remain static. There's no logic for handling updates, deletions, or evolving model versions, which are essential for long-term scalability and correctness.

- **No handling of missing features:** The current implementation assumes that all ECGs have complete data across all simulated feature groups. In real-world scenarios, however, feature availability often varies (for example, certain ML models may only apply to specific ECGs).

4.1 Trade-offs

- **Accuracy vs. efficiency in index selection**: `IndexFlatL2` provides exact results but scales poorly with data size, while `IndexHNSWFlat` offers faster approximate retrieval, especially useful for high-dimensional features. The efSearch parameter controls the recall–speed trade-off and can be tuned as needed.

- **Feature granularity vs. efficiency:** Aggregating related features (e.g., `risk_*` into `risk_scores`) improves performance and simplifies logic, but reduces the ability to emphasize individual components in similarity queries.

- **Speed vs. fidelity in dimensionality reduction:** PCA reduces index size and improves speed for embeddings but may eliminate useful variance, slightly reducing the expressiveness of the similarity metric.

- **Scalability vs. robustness in candidate expansion:** Using a `margin_factor` boosts recall by retrieving more candidates per group but adds memory and time costs during aggregation.

- **System simplicity vs. flexibility:** Supporting modular queries and per-group control improves interpretability and customizability, but increases memory usage and implementation complexity.

5. Future work and production considerations

5.1 Future work

- **Tuning HNSW and aggregation parameters**: While parameters like efSearch, M, and margin_factor were held constant in our experiments, their tuning could substantially improve recall-speed trade-offs. Systematic benchmarking would allow dynamic adjustment based on query context and available resources.

- **Expanded feature set**: The current design supports only four feature groups. Future iterations could incorporate waveform-derived intervals (e.g., QT, PR), demographic metadata, and condition-specific model outputs to enrich the retrieval space.

- **Adaptive weighting based on model reliability**: We did not tune feature weights during our experiments. Incorporating model performance indicators, coming from performance or uncertainty, into the weighting logic, could improve the robustness and interpretability of similarity scores, ensuring that more reliable metrics have greater influence in the search.

5.2 Production considerations

- **Compression via quantization**: FAISS offers **compressed index types** like `IndexIVFPQ` or `IndexHNSWPQ` that combine product quantization with approximate search. These reduce memory usage significantly while preserving acceptable accuracy [8, 10, 13].

- **GPU acceleration**: FAISS supports **CUDA-based GPU execution**, which can accelerate both training and querying of large indexes — particularly helpful for high-dimensional embeddings and fast response times in production [8, 10, 13, 14].

- **Distributed indexing**: FAISS can be extended to **multi-node deployments** using **Distributed FAISS**, which allows sharding and querying across machines in parallel, a must for billion-scale workloads [13, 15].

- **Monitoring and updating**: A production system must track index drift and support incremental updates as new ECGs are ingested. This includes retraining of embedding models, reindexing, and invalidation of stale entries.
- **Metadata management and traceability**: Clinical use cases require traceability. Versioning of features, model checkpoints, commit hashes, timestamps, etc. should be integrated for safety and reproducibility.

6. Codebase and reproducibility

The full implementation is provided in the github repository `ecg-similarity-engine`. The codebase includes:

- `notebooks/ecg_similarity_search_poc.ipynb`: Main notebook containing the proof-of-concept, demonstrating the full pipeline (synthetic ECG generation, feature preprocessing, indexing, and similarity querying).
- `notebooks/ecg_similarity_search_benchmarking.ipynb`: A secondary, exploratory notebook used to benchmark query performance across different dataset sizes, metric groupings, and top-k values.
- Modular scripts (`src/`): `data_generator.py`, `data_preprocessor.py`, `similarity_searcher.py`, `hybrid_indexer.py`, etc. These implement the core logic of data simulation, preprocessing, indexing, and querying with FAISS.

See the `README.md` for a complete guide on installing dependencies and executing the notebooks.

7. References

[1] M. Nakayama, R. Yagi, and S. Goto, "Deep Learning Applications in 12-lead Electrocardiogram and Echocardiogram," *JMA Journal*, vol. 8, no. 1, pp. 102–112, 2025, doi: 10.31662/jmaj.2024-0195.

[2] G. Silva, P. Silva, G. Moreira, V. Freitas, J. Gertrudes, and E. Luz, "A Systematic Review of ECG Arrhythmia Classification: Adherence to Standards, Fair Evaluation, and Embedded Feasibility," arXiv.org. [Online]. Available: https://arxiv.org/abs/2503.07276

[3] S. Tahery, F. H. Akhlaghi, and T. Amirsoleimani, "HeartBERT: A Self-Supervised ECG Embedding Model for Efficient and Effective Medical Signal Analysis," arXiv.org. [Online]. Available: https://arxiv.org/abs/2411.11896

[4] M. Kull, T. M. S. Filho, and P. Flach, "Beyond sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration," *Electronic Journal of Statistics*, vol. 11, no. 2, pp. 5052–5080, Jan. 2017, doi: 10.1214/17-EJS1338SI.

[5] P. Rujirachun, P. Wattanachayakul, P. Phichitnitikorn, N. Charoenngam, J. Kewcharoen, and A. Winijkul, “Association of premature ventricular complexes and risk of ischemic stroke: A systematic review and meta-analysis - PMC,” *Clinical Cardiology*, vol. 44, no. 2, doi: 10.1002/clc.23531.

[6] A. L. W. Shroyer, J. F. Collins, and F. L. Grover, “Evaluating Clinical Applicability: The STICH Trial’s Findings,” *Journal of the American College of Cardiology*, vol. 56, no. 6, pp. 508–509, 2010.

[7] M. Yang *et al.*, “Excessive Supraventricular Ectopic Activity and the Risk of Atrial Fibrillation and Stroke: A Systematic Review and Meta-Analysis,” *Journal of Cardiovascular Development and Disease*, vol. 9, no. 12, Dec. 2022, doi: 10.3390/jcdd9120461.

[8] M. Douze *et al.*, “The Faiss library,” arXiv.org. [Online]. Available: https://arxiv.org/abs/2401.08281

[9] “Electrocardiogram Report Generation and Question Answering via Retrieval-Augmented Self-Supervised Modeling.” [Online]. Available: https://arxiv.org/html/2409.08788v1

[10] “Introduction to Facebook AI Similarity Search (Faiss),” Pinecone. Accessed: Jun. 18, 2025. [Online]. Available: https://www.pinecone.io/learn/series/faiss/faiss-tutorial/

[11] “What is the typical time complexity of popular ANN (Approximate Nearest Neighbor) search algorithms, and how does this complexity translate to practical search speed as the dataset grows?” Accessed: Jun. 20, 2025. [Online]. Available: https://milvus.io/ai-quick-reference/what-is-the-typical-time-complexity-of-popular-ann-approximate-nearest-neighbor-search-algorithms-and-how-does-this-complexity-translate-to-practical-search-speed-as-the-dataset-grows?

[12] Y. A. Malkov and D. A. Yashunin, “Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs,” in *IEEE Xplore*, Apr. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8594636

[13] D. Bajaj, “Scaling Semantic Search with FAISS: Challenges and Solutions for Billion-Scale Datasets,” *Medium*, Dec. 25, 2024. Accessed: Jun. 20, 2025. [Online]. Available: https://medium.com/@deveshbajaj59/scaling-semantic-search-with-faiss-challenges-and-solutions-for-billion-scale-datasets-1cacb6f87f95

[14] facebookresearch, “Faiss on the GPU,” GitHub. Accessed: Jun. 20, 2025. [Online]. Available: https://github.com/facebookresearch/faiss/wiki/FAISS-on-the-GPU

[15] facebookresearch, “GitHub - facebookresearch/distributed-faiss: A library for building and serving multi-node distributed faiss indices.,” GitHub. Accessed: Jun. 20, 2025. [Online]. Available: https://github.com/facebookresearch/distributed-faiss