An Airflow-Orchestrated AI Pipeline for Podcast Transcription, Topic Modeling, and Recommendation System

Ioannis Kazlaris *, Georgios Papadopoulos *, Konstantinos Diamantaras, Marina Delianidi, Eftychia Touliou, Anagnostis Yenitzes

Posted Date: 7 November 2025

doi: 10.20944/preprints202511.0449.v1

Keywords: podcast; AI-pipeline; transcription; topic modeling; recommendation system; airflow

 doi:10.20944/preprints202511.0449.v1

Article

An Airflow-Orchestrated AI Pipeline for Podcast Transcription, Topic Modeling, and Recommendation System

Ioannis Kazlaris [1,*], Georgios Papadopoulos [2,*], Konstantinos Diamantaras [1], Marina Delianidi [1], Eftychia Touliou [2] and Anagnostis Yenitzes [2]

[1] International Hellenic University
[2] European School Radio
* Correspondence: ikazlaris@ihu.gr (I.K.); gpapas@europeanschoolradio.eu (G.P.)

Abstract

This study presents a production-ready AI pipeline for audio content processing, implemented within the Youth Radio platform (www.youthradio.eu), which serves as an extension of the European School Radio initiative (www.europeanschoolradio.eu). The system uses a multi-server architecture: an AI Server that runs batch/offline jobs, orchestrated by Apache Airflow, and two Web Servers that deliver all the Backend as well as the Frontend applications, configured with load balancing and redundancy to ensure high availability and fault tolerance. The implemented AI-Pipeline includes tasks such as preprocessing, transcription, audio classification and topic modelling. Processed Podcasts are indexed in a Qdrant vector database to facilitate both dense and sparse retrieval while a recommendation system enriches the user's experience. We summarize design choices and report system-level metrics and task-level indicators (ASR quality after correction, retrieval effectiveness) to guide similar deployments.

Keywords: podcast; AI-pipeline; transcription; topic modeling; recommendation system; airflow

1. Introduction

1.1. Background and Motivation

Podcasts have emerged as a dominant medium for information sharing, entertainment, and education, offering rich and diverse content across languages, formats, and styles. However, their inherently unstructured nature poses significant challenges for discoverability, retrieval, and personalized recommendation. Unlike textual data, podcasts are stored as audio files, requiring complex processing pipelines to extract, structure, and enrich content in ways that make it accessible for search, categorization, and user personalization. Recent advances in artificial intelligence — including automatic speech recognition (ASR), topic modeling, and embedding-based retrieval — have enabled the large-scale transformation of audio into structured, searchable, and semantically meaningful representations. Yet, operationalizing these capabilities in a scalable and maintainable production environment remains a non-trivial task. The integration of multimodal AI systems that can process not only speech content but also contextual metadata, speaker characteristics, and temporal dynamics presents additional architectural complexities that must be carefully balanced against computational costs and real-time performance requirements. Furthermore, the challenge is compounded by the need to handle diverse audio quality conditions, multiple languages, domain-specific terminology, and evolving user preferences while maintaining consistent accuracy and relevance across millions of podcast episodes. These complexities manifest in several specific technical dimensions when building such systems at scale.

 doi:10.20944/preprints202511.0449.v1

1.2. Challenges

Building an end-to-end AI pipeline for podcasts involves multiple technical challenges:

- Heterogeneous Audio Data: Varying recording quality, background noise, poorly recorded audio, and multilingual content require robust preprocessing and adaptable models.
- Large-Scale Processing: With thousands of hours of content, the system must support high-throughput, GPU-intensive inference without disrupting ongoing user services.
- Complex Orchestration: Coordinating multiple AI models, each with distinct runtime requirements, demands reliable task scheduling, monitoring, and failure recovery.
- Computational Workload Management: Efficiently distributing computational workloads between front-facing services and heavy inference tasks is essential for performance and scalability.
- Personalization & Retrieval Accuracy: Generating accurate recommendations and search results depends on effective integration of metadata filtering, semantic embeddings, and user preference modeling.

1.3. Scope of This Work

We describe an end-to-end use case of a podcast processing platform that currently handles over 11,000 podcasts, with the number continuously growing, built on a multi-server design where Apache Airflow orchestrates the batch/offline processing. The system ingests raw podcast audio in various formats, transcribes speech with a fine-tuned multilingual ASR model, followed by context-dependent grammar correction. In addition, it applies audio classification and topic modeling on the transcribed text to derive the thematic categories of the podcasts. The enriched dataset powers both a hybrid search engine—based on embeddings and metadata—and a personalized recommendation system. By separating web-related services from AI inference workloads, our architecture ensures both scalability and operational stability.

1.4. Contributions

The key contributions of our work are as follows:

- End-to-End AI Podcast Pipeline: An integrated framework combining audio preprocessing, ASR, audio classification, topic modeling, semantic retrieval, and recommendation into a single, unified workflow.
- Multi-Server Deployment: A separation of web services and inference tasks, enabling efficient GPU utilization and uninterrupted user access.
- Orchestration with Apache Airflow: Automated, fault-tolerant scheduling and coordination of processing tasks, ensuring scalability and maintainability.
- Dense Retrieval and Sparse Retrieval: Integration of dense semantic retrieval with metadata filtering to improve both relevance and personalization.
- Recommendation System: Generating accurate recommendations and search results depends on effective integration of metadata filtering, semantic embeddings, and user preference modeling.
- Real-World Implementation: Deployment and evaluation on a large corpus of multilingual podcasts, demonstrating the practicality of the proposed system.

2. Related Works

Recent work has converged on end-to-end pipelines that transform long-form podcasts into searchable, recommendable objects. The TREC 2020 Podcasts Track catalyzed this area by defining segment retrieval and summarization tasks on a 100K-episode corpus, benchmarking deep models for both search and summarization. A companion perspective outlines domain challenges such as noisy ASR, support for multiple languages, and evaluation [1]. Architecturally, systems lean on segmentation and topic structure: dynamic or learned segmentation improves retrieval efficiency [2], and multimodal topic segmentation sharpens navigation within shows [3]. For summarization,

 doi:10.20944/preprints202511.0449.v1

competitive TREC submissions demonstrate pipeline designs that first select salient spans, then apply sequence-to-sequence generators, validating the value of staged processing for long transcripts [4].

Datasets and representations underpin these pipelines. The Spotify Podcast Dataset and the 100,000 Podcasts corpus provide large-scale audio and transcript resources that drive IR, summarization, and dialog modeling research [5,6]. Feature-level work precomputes audio descriptors to accelerate experimentation and hybrid (audio and text) retrieval [7]. New tasks expand user-facing utility: creation of chapters (PODTILE) uses transformer models to generate boundaries and titles that improve browsing and even downstream search, aligning closely with your pipeline's "structure-then-retrieve" philosophy [8]. User-study infrastructure (Podify) supports logging and evaluation for recommendation and interaction research [9]. Language-specific corpora (e.g., Greek Podcast Corpus) demonstrates that weakly supervised podcast data can materially improve ASR for under-resourced languages—useful when the system targets Greek content [10].

For automation, Airflow emerges both as an execution substrate and as a reproducibility layer. CWL-Airflow bridges standard workflow specs (CWL) with Airflow's scheduler/executor, highlighting portability from laptop to cluster/cloud [11]. Empirical studies of "workflows-as-code" in Airflow identify failure modes (configuration, environment complexity) and best practices relevant to resilient DAGs in production [12]. Broader WfMS comparisons (bioinformatics & scientific workflows) offer criteria—expressiveness, modularity, scalability, robustness—to motivate Airflow in data/ML pipelines while clarifying trade-offs with alternative engines [13].

3. System Overview

Figure 1 illustrates the production deployment used by Youth Radio (www.youthradio.eu) and European School Radio (www.europeanschoolradio.eu). The architecture separates real-time user services from GPU-intensive AI workloads through a multi-server design.

- Web Servers host the user interface, search and browse features, and REST APIs for authentication, CRUD operations, and metadata filtering. They manage podcast metadata (MySQL and MongoDB) and caching (Redis) to ensure fast user interaction.
- AI Server runs the podcast processing pipeline as Apache Airflow DAGs. This includes speech-to-text transcription (FasterWhisperXXL) [14], post-ASR correction (Gemini 2.5 Flash) [15], audio classification (BEATs) [16], topic modeling (BERTopic) [17], topic classification via text similarity (MPNet) [18], embedding generation (LaBSE) [19], and recommendation system. Vector data is stored in Qdrant, while job orchestration relies on a MySQL queue.
- Network Attached Storage (NAS) provides centralized object storage for podcasts, user, and school data.

Airflow coordinates ingestion and long-running tasks, ensuring reliability with retries and monitoring. By decoupling user-facing services from batch processing, the system maintains low interactive latency while supporting continuous enrichment of podcast data for semantic search and personalized recommendation.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2025 doi:10.20944/preprints202511.0449.v1

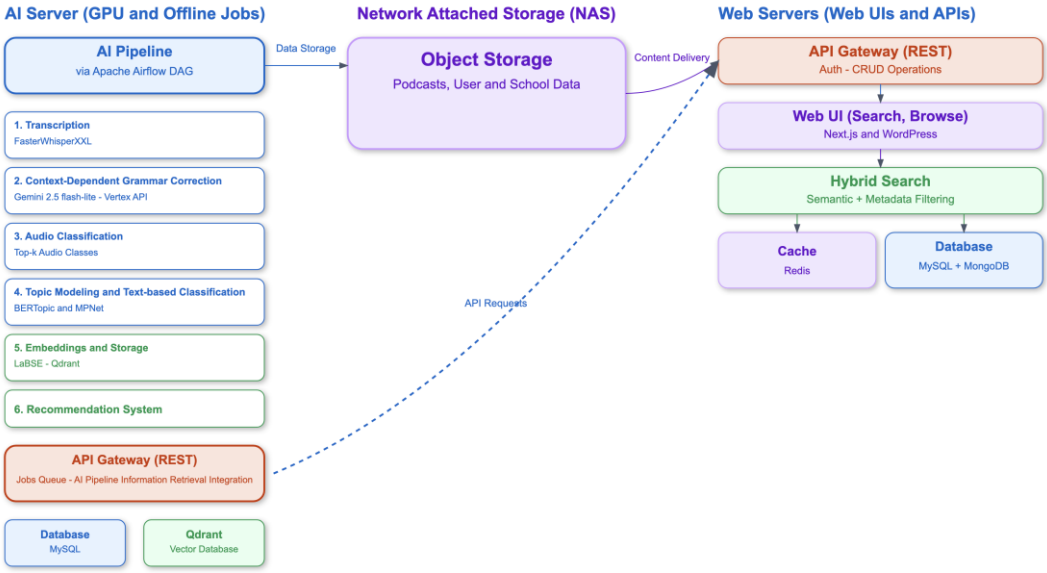


Figure 1. Overview of the implemented multi-server architecture, with discreet roles for the AI pipeline and web-related services.

To streamline integration with our AI pipeline, we used MySQL to inspect and extract the internal schema of our podcast database. We then carefully simplified this structure, retaining only the fields necessary for processing and downstream AI tasks to ensure efficient data flow and minimize redundancy. This simplified structure is presented below:

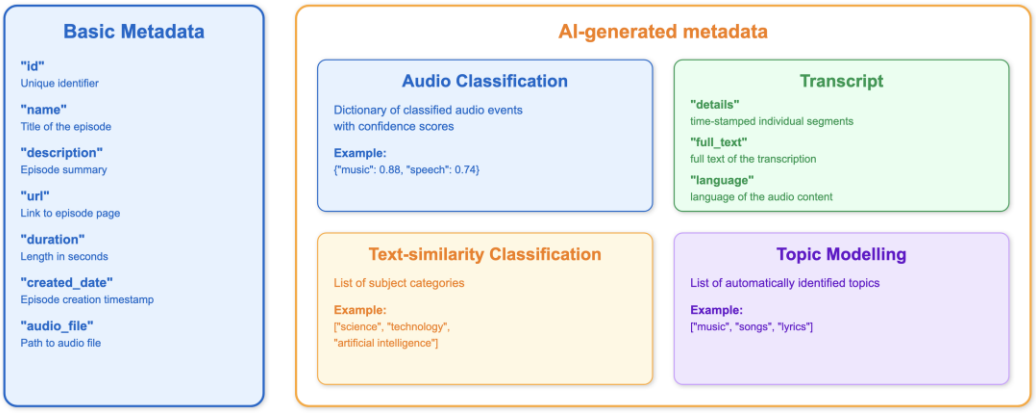


Figure 2. Basic metadata are directly extracted using MySQL whereas "audio_classes", "subjects", "transcript" and "topics" are populated by AI models.

4. AI Pipeline

4.1. Transcription

Transcription quality constitutes the foundation of our podcast processing pipeline, as all subsequent components—such as topic modeling, classification, and retrieval—rely directly on the accuracy and consistency of the automatic speech recognition (ASR) output. Even minor transcription errors can propagate through the system, affecting topic inference and semantic embedding generation. To ensure high transcription quality, we evaluated multiple model variants under identical conditions and selected FasterWhisperXXL [14]; all evaluated models are derived from OpenAI's Whisper architecture [20]. The evaluation was conducted on an NVIDIA A16 GPU with 16 GB of memory, which imposed practical limitations on model size, batch configuration, and inference throughput. We adopted small batch sizes to avoid kernel crashes when processing lengthy podcast

 doi:10.20944/preprints202511.0449.v1

episodes. Human evaluators assessed transcription quality, focusing on accuracy, naturalness, and handling of multilingual segments.

The comparative results are summarized in Table 1. Larger models generally improved transcription accuracy and human-rated quality but required more GPU memory and exhibited higher latency. Whisper Turbo achieved the fastest inference but produced less consistent results in spontaneous and multilingual speech. FasterWhisperXXL provided the best balance between transcription accuracy, speed, and resource efficiency, achieving the highest human-rated quality.

Table 1. Comparison of Whisper model variants based on transcription accuracy, latency, and GPU memory usage. Human-rated quality served as the final selection criterion.

Model	WER (%) ↓	Human-rated Quality (1–5)	Average Latency (× real-time)	GPU Memory Usage (GB)
Whisper Medium	17.8	3.3	0.85×	7.2
Whisper Large-v2	14.2	3.8	1.10×	10.8
Whisper Large-v3	12.6	4.1	1.25×	12.6
Whisper S2T	13.5	3.9	1.05×	9.4
Whisper Turbo	15.1	3.6	0.55×	8.3
FasterWhisperXXL	11.0	4.3	0.90×	11.5

Table 1 summarizes the trade-offs between speed, accuracy, and resource efficiency, guiding our final selection of FasterWhisperXXL. However, we also encountered a recurring issue: the model frequently attempted to transcribe lyrics from background music embedded in podcast episodes. This behavior was particularly problematic in multilingual contexts, where English-language songs were interspersed within non-English dialogue. To mitigate this, we explicitly guided the model using a system prompt instructing it to "transcribe the audio as accurately as possible, to disregard any music, lyrics, or background noises, and to avoid translating English or any other foreign languages."

Despite this targeted prompt, the model continued to transcribe lyrics and sometimes misinterpreted background sounds as meaningful speech. To address this more effectively, we disabled the model's automatic language detection feature and instead hard-coded the target language code directly into the inference pipeline. This adjustment significantly improved transcription fidelity, especially in podcasts where music and multilingual content coexist. We attribute this improvement to the fact that automatic language detection, when faced with strong musical or English-language signals, may shift the model's decoding behavior away from the desired linguistic context—even when contradicted by the prompt. These findings underscore a broader challenge in the current generation of AI models: their inconsistent adherence to prompt-based instructions. This limitation is not unique to Whisper or speech-to-text systems but is also widely recognized in large language models more generally. The underlying causes are multifaceted and may include issues related to the quality and distribution of pretraining data, limitations in prompt conditioning during fine-tuning, and model hallucinations triggered by ambiguous or noisy inputs.

As mentioned earlier, this stage of our AI pipeline is designed to handle podcast audio files by passing them through FasterWhisperXXL, which performs end-to-end transcription and outputs timestamped text segments. These transcriptions are structured and aligned with their corresponding audio positions, enabling fine-grained analysis, searchability, and topic segmentation later in the pipeline. We synthesize the final transcription by removing the timestamps of the individual segments and simply "stitching" together the text chunks.

4.2. Context-Dependent Grammar Correction

Error correction (EC) models are critical for improving Automatic Speech Recognition (ASR) transcriptions by boosting readability and quality. Without needing access to the code or model

 doi:10.20944/preprints202511.0449.v1

weights, EC can enhance performance and adapt black-box ASR systems to specific domains [21]. We use Google's API to call the Gemini 2.5 flash model via the Vertex AI API as a post-ASR correction step that is explicitly context-aware [15]. Unlike purely lexical or rule-based tools, Gemini conditions each correction on surrounding sentences so it can fix grammar, agreement, punctuation, and obvious typos without flattening meaning or overwriting proper nouns. In practice, this yields cleaner transcripts for downstream topic classification, embedding generation, and search, especially in noisy segments where FasterWhisperXXL's raw outputs contain localized errors that only make sense when viewed in context.

Implementation-wise, we send all the JSON Array of the generated timed transcript of the podcast via the Google's API with a tight prompt:

"You are a specialized editor for correcting ASR-generated transcripts.

Rules:

1. Correct spelling, grammar, punctuation, and capitalization.

2. Remove unnecessary word repetitions.

3. Fix incorrect word merges/splits (e.g. 'otherone' -> 'other one').

4. Replace unintelligible words with the closest phonetically/visually similar word, or otherwise with the most contextually appropriate one.

5. Preserve the original meaning and style exactly. Do not add new information. Do not change the word order.

6. Keep in mind that these subtitles were created by students, so they may follow contemporary trends in terminology.

7. The original format is JSON with subtitles. You must only edit the 'text' tag and return the results in the same format."

This prompt ensures that Gemini 2.5 flash-lite acts as a conservative editor rather than a creative re-writer, making only targeted corrections that improve readability while preserving meaning.

We evaluate Gemini's effect on transcript quality by comparing the corrected output against both FasterWhisperXXL's raw output and a manually created GOLD reference. Our analysis focuses on Word Error Rate (WER) [22] and Character Error Rate (CER) [23], which are widely used objective measures for transcription quality. In addition to headline results, we provide detailed diagnostic plots (scatter plots, CER distributions, and pairwise error differences) in Appendix A to support reproducibility and allow readers to explore the error patterns more deeply.

In Figure 3, the left panel shows FasterWhisperXXL and Gemini 2.5 flash-lite word error rates (WER) side by side, clearly indicating that Gemini 2.5 flash-lite generally yields slightly lower WER for most files (tighter distribution and lower median). The right panel overlays the two distributions, highlighting where the models agree and where differences occur. The bottom-heavy nature of both violins suggests that most transcriptions have relatively low WER (<0.2), but Gemini shows a subtle shift toward even lower values, confirming its role as a corrective model. Outliers with higher WER remain in both systems, suggesting that some files are intrinsically challenging regardless of model choice.

 doi:10.20944/preprints202511.0449.v1

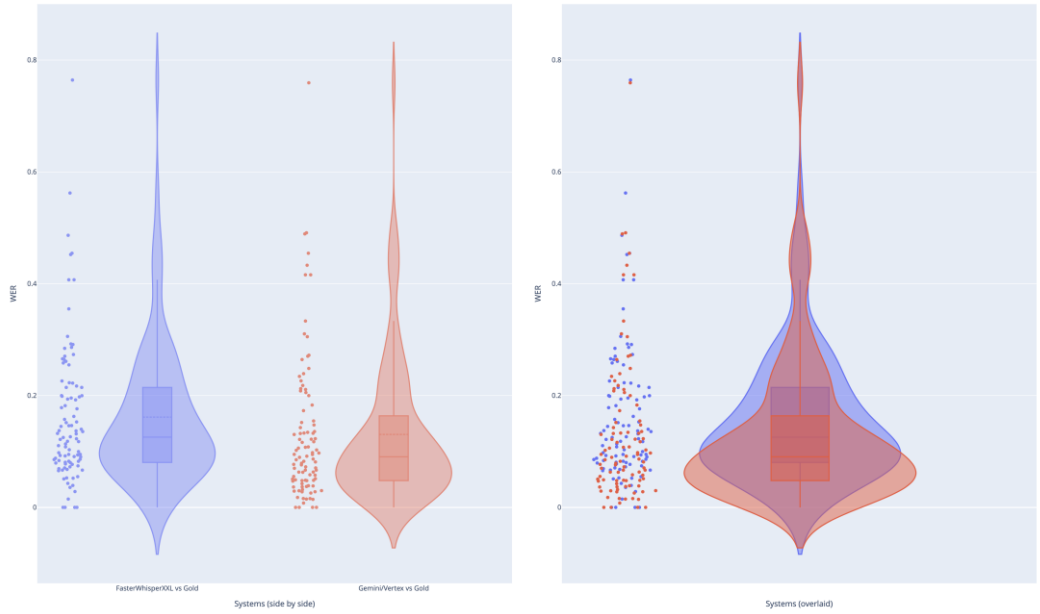


Figure 3. Comparison of Word Error Rates (WER) between FasterWhisperXXL and Gemini 2.5 flash-lite against the GOLD reference.

Figure 4 shows the distribution of Word Error Rate (WER) across 10%-wide bins for both FasterWhisperXXL and Gemini 2.5 flash-lite, compared against the GOLD reference. The largest concentration of podcasts falls in the 0–10% WER range for both models, but Gemini has noticeably more samples here (55% vs. 41%), suggesting that it moves many moderately erroneous transcripts into a near-perfect range. Conversely, FasterWhisperXXL has more samples in the 10–30% range, showing that its output is slightly less polished on average. Both systems have very few samples above 50% WER, confirming that most podcasts are reasonably well-transcribed.

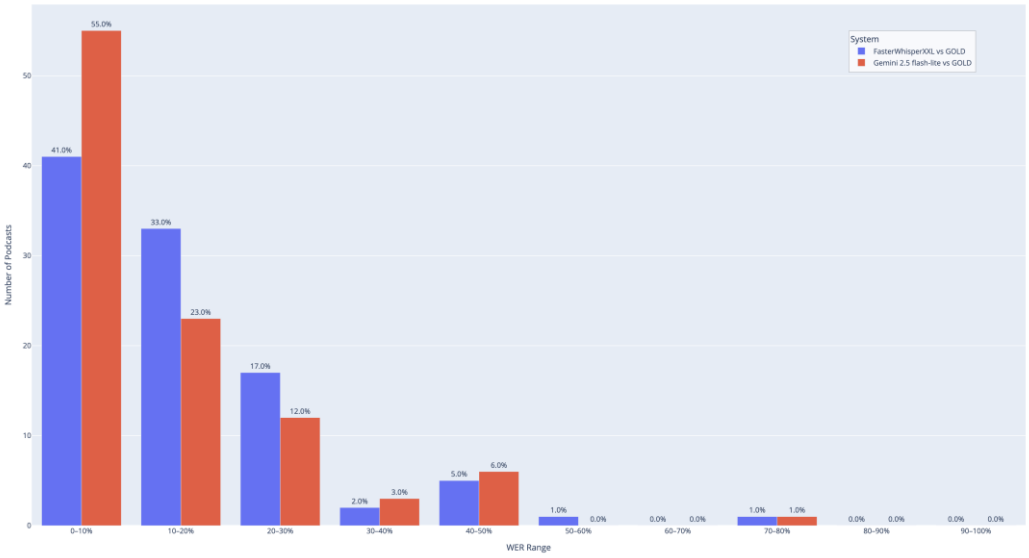


Figure 4. Distribution of WER across 10%-wide bins for FasterWhisperXXL and Gemini 2.5 flash-lite.

The ECDF plot in Figure 5 shows the cumulative fraction of podcasts below each WER value. Gemini 2.5 flash-lite (red) consistently stays above FasterWhisperXXL (blue) for most of the range, indicating a larger share of podcasts with lower WER. At the 10% WER mark, 55% of Gemini transcriptions fall below this threshold compared to 41% for FasterWhisperXXL — a meaningful

 doi:10.20944/preprints202511.0449.v1

improvement. The gap narrows beyond 0.3 WER, suggesting that Gemini's advantage is most pronounced for easier-to-moderate cases, with both systems performing similarly on more challenging audio.

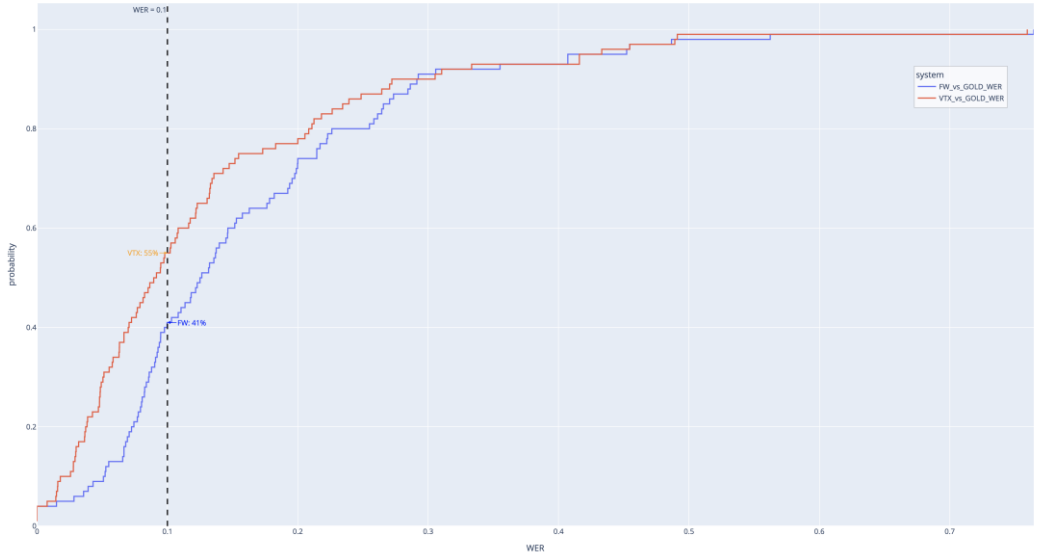


Figure 5. Empirical cumulative distribution function (ECDF) of WER for FasterWhisperXXL and Gemini 2.5 flash-lite against the GOLD reference.

4.3. Audio Classification

To enrich podcast metadata with sound-based context, our system integrates an advanced audio classification component powered by Microsoft's BEATs model [16]. BEATs (Bootstrap your Own Audio Transformer with Self-supervised Learning) is a state-of-the-art transformer-based architecture that learns robust audio representations through large-scale self-supervised pretraining. It has demonstrated strong performance on a range of downstream audio tasks, particularly classification benchmarks such as AudioSet, which includes over 500 predefined sound event categories.

Leveraging BEATs' fine-tuning capabilities, we apply the model to classify audio segments from podcasts into meaningful sound classes—such as speech, music, applause, laughter, environmental sounds, and various types of instrumentation. This classification provides a layer of non-verbal context that complements textual transcriptions. This approach is particularly valuable in educational, thematic, or entertainment-driven podcast discovery scenarios. By incorporating this auditory dimension, users gain the ability to search for or filter podcasts not just by what is said, but also by how it sounds.

To implement this functionality, we created a custom CSV file that maps BEATs' internal class identifiers to human-readable labels derived from the AudioSet ontology. This mapping enables transparent interpretation of the model's predictions. The predicted class probabilities are parsed and stored within the podcast's metadata structure, allowing downstream applications to access and visualize the most salient sound categories.

Figure 6 illustrates the top 10 audio event classes across our podcast collection, ranked by their average confidence scores. These dominant acoustic signatures help reveal latent patterns thus supporting both analytic use cases (e.g., sound-rich content clustering) and user-facing features (e.g., "search by sound").

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2025 doi:10.20944/preprints202511.0449.v1

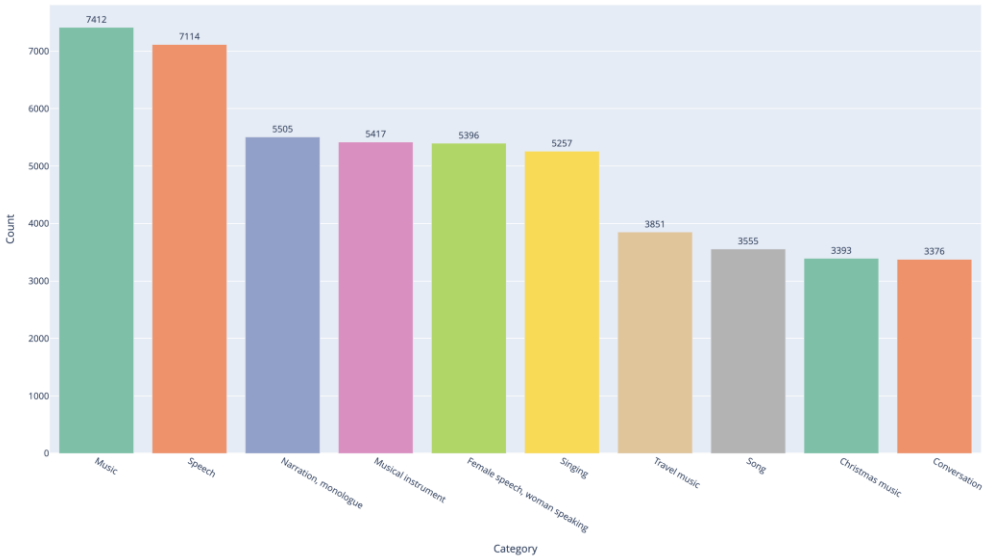


Figure 6. Top 10 audio event classes across our podcast collection.

As illustrated in Figure 7, the t-SNE projection reveals the semantic structure of the podcast corpus, where each point represents a podcast segment positioned according to its high-dimensional audio embedding. This representation offers an intuitive, high-level overview that facilitates downstream tasks such as content filtering, clustering, personalized recommendation, and interactive exploration. The distribution highlights that the dominant acoustic classes are Music and Speech while smaller but distinct clusters indicate the presence of more specialized acoustic events.

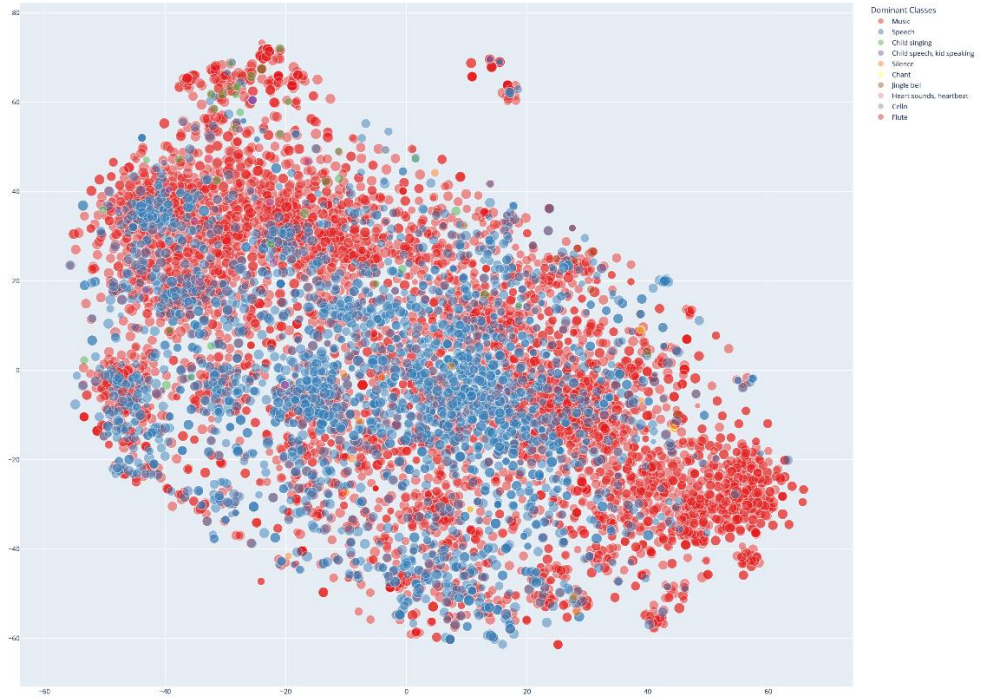


Figure 7. Top 10 audio classes with semantic overview over our vector database.

4.4. Topic Modeling

To provide thematic navigation and organization of podcasts, our system employs BERTopic—a transformer-based topic modeling framework [17]. BERTopic identifies latent topics within

 doi:10.20944/preprints202511.0449.v1

transcribed text, assigning probabilistic topic distributions to each podcast. This allows the system to label and group podcasts by subject matter without requiring predefined taxonomies. The extracted topics not only enhance discoverability but also improve content filtering and browsing. Users can therefore explore educational content through intuitive categories while BERTopic's dynamic clustering allows the system to adapt as new podcast themes emerge over time.

To ensure high-quality, multilinguality and semantically rich embeddings tailored to our data, we selected the LaBSE model from the Sentence Transformers library as our text encoder [19]. This model is particularly well-suited to our dataset and offers state-of-the-art performance in sentence-level semantic similarity tasks. For clustering, we utilized BERTopic's default density-based algorithm, HDBSCAN, which is known for its ability to discover clusters of varying density and shape in high-dimensional spaces. We configured it with a min_cluster_size of 5 and used the "euclidean" distance metric, which provided stable and interpretable clusters in our embedding space. In terms of vectorization, we retained the default CountVectorizer and class-based TF-IDF (CTF-IDF) Transformer, both of which performed adequately for our needs without requiring additional tuning.

One challenge we encountered was the tendency of BERTopic to generate an excessively granular set of initial topics, especially when dealing with large or diverse transcript datasets. To address this, we conducted a series of experiments with the nr_topics parameter, which allows for topic reduction and merging after the initial clustering step. Through empirical evaluation, we found that setting nr_topics=50 yielded a more manageable and semantically coherent topic set, striking a balance between thematic granularity and interpretability.

The podcast metadata is first loaded into a Pandas DataFrame, where each transcript serves as the primary input for topic extraction. Prior to topic modeling, we implemented an additional preprocessing stage that scans and sanitizes the transcripts by removing uncommon Unicode symbols, non-printable control characters, and Private Use Area (PUA) codepoints. This step ensures textual consistency and prevents corrupted or noisy characters from influencing the embedding and clustering processes. Once preprocessed, the cleaned transcripts are passed to the BERTopic model, which assigns a dominant topic label and a corresponding probability distribution to each podcast. This probabilistic output enables fine-grained thematic indexing and confidence-aware classification of content.

4.5. Topic Classification via Text Similarity

For podcasts requiring assignment to a predefined set of categories, we employ a training-free, similarity-based pipeline built on the multilingual paraphrase-multilingual-mpnet-base-v2 encoder [18]. This component complements BERTopic's unsupervised clustering by enforcing consistent, taxonomy-aligned labels while remaining lightweight and easy to maintain for multilingual educational content. Concretely, each category is encoded and each podcast transcript is chunked on sentence boundaries. We then embed label prompts and transcript chunks with MPNet, compute cosine similarities across all chunk–label pairs, and aggregate per label (max or mean over chunks) to produce a ranked top-k shortlist (default k = 3). Because the method is training-free, topic descriptions can be periodically updated (e.g., after a fixed number of new podcasts or at regular intervals) without retraining; otherwise, the same encoder classifies new items efficiently.

The candidate label set—e.g., "Ειρήνη και Μη Βία", "Περιβάλλον, Κλίμα, Βιωσιμότητα", "Βιντεοπαιχνίδια, Gaming"—originates from earlier BERTopic exploration and is "humanized" for interpretability and relevance to Youth Radio and European School Radio. This similarity-first design is robust to implicitly discussed themes and supports search, recommendation, and curriculum-aligned tagging as the taxonomy evolves. Evaluation (top-k, ranking-aware). On 47 categories (N = 100), the classifier attains Top-1 accuracy = 0.39, Hit@3 = 0.68, P@3/R@3/F1@3 = 0.307, MRR = 0.617, and Jaccard@3 = 0.212. We emphasize ranking metrics because the application surfaces a shortlist to users; In contrast, full multilabel set-matching metrics (micro/macro P/R/F1 over all 47 labels) assume unordered, exhaustively annotated label sets and a globally thresholded multi-label classifier—

 doi:10.20944/preprints202511.0449.v1

assumptions that do not fit a ranked top-k, similarity-based system. For each podcast, the similarity-based ranker compares transcript chunks to label prompts and returns a top-k shortlist, enabling assignment to one or more semantically appropriate topics without supervised retraining. This pipeline consistently categorizes new content against the educational taxonomy, supporting accurate filtering, discovery, and content tagging across diverse episodes and series.

4.6. Embedding Generation & Retrieval

To enable efficient semantic search and content matching, our system uses Qdrant as the vector database for dense retrieval. Podcast transcripts are encoded with LaBSE [19]; the resulting embeddings are L2-normalized and ranked by cosine similarity, then stored in Qdrant alongside descriptive metadata from the original MySQL database. User queries (or embeddings derived from listening history) are matched via approximate nearest-neighbor search, while hybrid retrieval applies metadata filters (e.g., topics, audio classes, language, recency) to refine results. Dense retrieval surpasses keyword matching by capturing semantic relatedness, so the platform surfaces podcasts that are thematically aligned—not just textually similar—to a user's intent. Qdrant's low-latency ANN search and payload filtering support real-time search and recommendations, and incremental upserts integrate cleanly with Airflow indexing to keep results fresh.

In addition to storing vector representations, we systematically integrate structured metadata from the relational MySQL database into Qdrant. Each podcast entry carries rich contextual attributes—such as title, description, generated transcription, generated subtitles, predicted labels and extracted audio classes—that are extracted and ingested as payloads alongside the embeddings. This dual storage strategy enables hybrid retrieval pipelines to combine semantic similarity with precise metadata filtering in a single query, ensuring that recommendations are not only thematically aligned but also contextually relevant. By bridging the structured world of relational databases with the unstructured domain of embeddings, the platform achieves both depth and precision in retrieval, while maintaining consistency across heterogeneous data sources.

Qdrant's retrieval efficiency is grounded in its implementation of Approximate Nearest Neighbor (ANN) search algorithms. Instead of scanning the entire embedding space exhaustively, Qdrant organizes vectors into optimized indexes (e.g., HNSW graphs) that drastically reduce lookup complexity. HNSW (Hierarchical Navigable Small World) graphs balance high recall with low latency, allowing queries to traverse only a fraction of the vector space while still returning results that closely approximate the true nearest neighbors. This design enables millisecond-level response times even as the collection grows to millions of vectors, ensuring scalability without compromising retrieval quality. To empirically validate these claims, we conducted a series of performance tests on Qdrant under increasing dataset sizes. The results of these experiments are presented in Figure 8.

 doi:10.20944/preprints202511.0449.v1

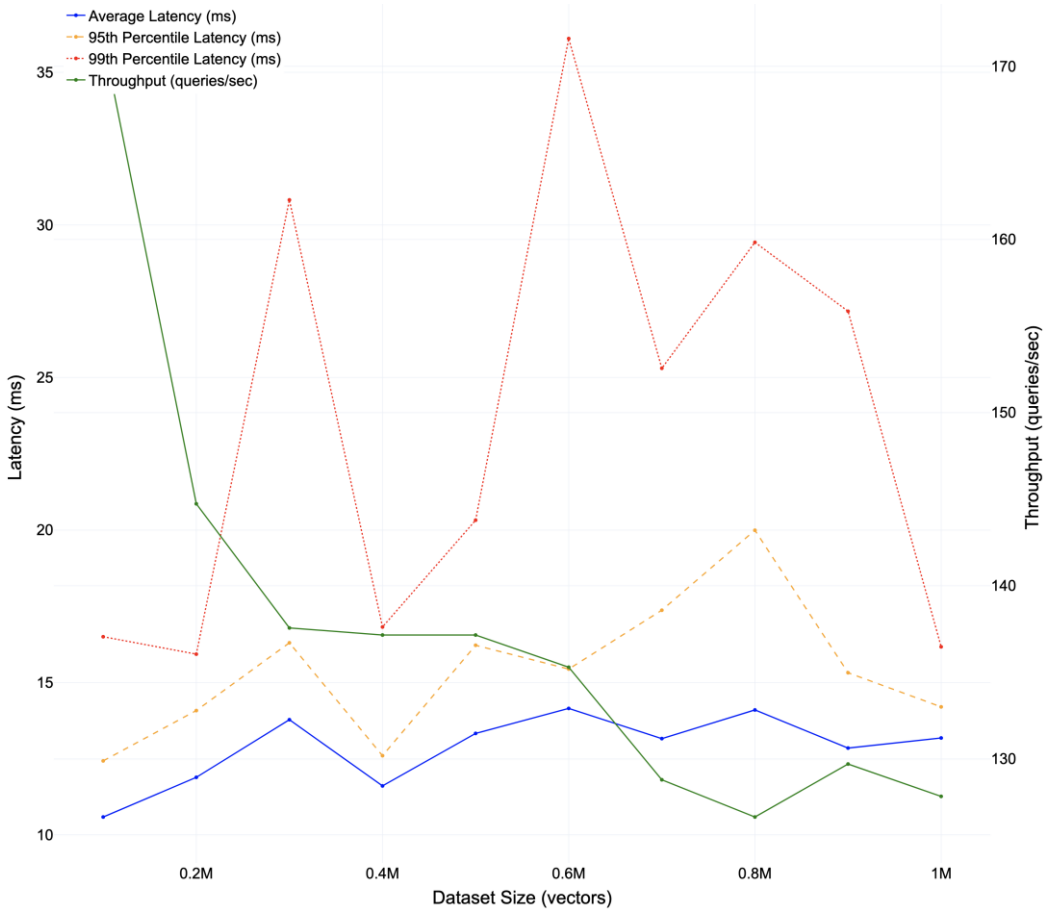


Figure 8. Qdrant performance benchmark across increasing dataset sizes. The plot reports average latency, 95th percentile latency, 99th percentile latency, and throughput, demonstrating that retrieval remains efficient and scalable even at one million vectors.

Figure 8 highlights that Qdrant sustains consistently low retrieval latency even as the vector collection scales to one million entries. The average latency remains well below 20 ms, with only moderate increases observed at the 95th and 99th percentiles, confirming the efficiency of HNSW-based indexing under production workloads. Throughput exhibits only a slight decline with increasing dataset size, indicating that the system preserves near real-time responsiveness and scalability—both essential for maintaining high-quality search and recommendation performance in a continuously growing podcast library.

Beyond improving retrieval speed, the integration of Qdrant in our system enables payload-aware filtering, allowing semantic search results to be refined using metadata ingested from MySQL. This capability allows our retrieval layer to combine semantic similarity with structured constraints—for instance, filtering podcasts by production date or category while ranking results by embedding proximity. As a result, search outputs are both semantically relevant and contextually precise. From an operational standpoint, Qdrant's incremental upserts, sharding, and replication mechanisms have proven effective in maintaining continuous index updates through our Airflow-based ingestion pipeline. New podcasts and metadata modifications can be integrated without service interruption, ensuring that the retrieval index remains consistently up to date. The system's distributed design supports fault tolerance and horizontal scalability, aligning with our production requirements for low-latency, high-availability retrieval in a dynamic content environment.

4.7. Recommendation System

The final stage of the AI pipeline is the recommendation engine, which suggests new podcasts tailored to individual user preferences. By combining dense similarity search with user history and

 doi:10.20944/preprints202511.0449.v1

preferences stored in the metadata database, the system dynamically adapts to user behavior. Recommendations improve over time, offering increasingly relevant and diverse content. This component fosters engagement by surfacing content that users might not explicitly search for but would likely enjoy based on past interactions. It also supports cross-podcast discovery by linking semantically related shows while providing a more interconnected content experience.

The core of this system is a vector store containing podcast embeddings, which are compared using cosine similarity against user-specific vectors derived from their listening history and preferences. User metadata is synchronized in real time: when a listener plays an episode or likes a podcast, these events trigger immediate ingestion through the same Airflow-driven pipelines that manage content metadata. The updated signals are propagated into Qdrant, where they are stored alongside embeddings. This event-level synchronization ensures that the recommendation engine adapts to user behavior, aligning podcast suggestions with the most recent listening activity. After performing similarity searches, our system retrieves the top-N most relevant podcasts. This recommendation logic is API-compatible, enabling seamless communication between the AI Server and the Web Server. It also supports dynamic filtering using tags or audio classes, creating a hybrid recommender that combines semantic similarity with metadata-driven constraints.

To optimize the quality and responsiveness of recommendations, we evaluated two distinct strategies for modeling user interests: The first one is based on discrete embedding aggregation while the second one in based on weighted average embedding.

1. Discrete Embedding Aggregation

In this approach (Table 2), the system maintains the individual embeddings of all podcasts that a user has listened to or favorited. These embeddings are used collectively to retrieve similar content from the vector space without collapsing them into a single representation. Each embedding serves as an independent probe into the vector space, allowing the system to:

- Capture Multi-faceted Interests: Users often have diverse, seemingly disparate interests that would be lost in averaging. For instance, a user might enjoy both true crime podcasts and meditation content - maintaining separate embeddings preserves both preference clusters.
- Preserve Semantic Granularity: Individual embeddings retain the specific semantic fingerprints of each podcast, including genre-specific terminology, thematic elements, and stylistic characteristics that contribute to the user's preference profile.
- Enable Dynamic Weighting: The system can apply sophisticated weighting schemes based on recency, explicit user ratings, listening duration, or engagement metrics without losing the underlying preference diversity.

Table 2. Top-k (k = 5) recommendations for a random user. Embeddings are not averaged. Recommendations are based on the user's listening history and preferences.

Model	Similarity ↓	Description
Climate! Can we change?	75.47%	The climate is changing throughout the world. We are the children of this world...
RADIO GIRLS-Climate change	73.78%	Climate change: what can we do...
Music Radioactivity	73.11%	A show dedicated to women... on the occasion of International Women's Day...
The climate...on your neck	71.70%	The climate change is already a reality. The temperature is increasing...
Women's Diaries	70.82%	Gender Equality, Sports, work, housekeeping and interpersonal relationships...

2. Weighted Average Embedding

 doi:10.20944/preprints202511.0449.v1

In the second version (Table 3) we compute a single, aggregated user embedding by taking a weighted average of the embeddings from their listening history and preferences. This representation is then used as a query vector during similarity search. The system constructs the composite user embedding through sophisticated vector arithmetic operations:

- Weighted Vector Summation: Individual podcast embeddings are multiplied by their respective weight coefficients, which may be derived from listening duration, explicit ratings, recency factors, or engagement frequency metrics.
- Normalization Operations: The weighted sum undergoes L2 normalization to maintain unit vector properties, ensuring consistent similarity computation across different user profiles regardless of their listening volume.
- Dimensional Consistency: The aggregation process preserves the original embedding dimensionality of the 768 dimensions, maintaining compatibility with the pre-trained podcast embedding model.

Table 3. Top-k (k = 5) recommendations for the same random user. Embeddings are averaged. Recommendations are based on the user's listening history and preferences.

Model	Similarity ↓	Description
M Squared	69.27%	In today's show we will talk about the movie Good Will Hunting...
Music Radioactivity	67.83%	A show dedicated to women... on the occasion of International Women's Day...
Women's Diaries	65.03%	The show is dedicated to women. Through the narration...
Voices of History	62.63%	The podcast “Voices of History” was created by our school’s radio team...
Women in education	59.45%	Interviews with important women in education...

Through empirical testing, we found that the first strategy—maintaining separate podcast embeddings per user—consistently yielded more precise and satisfying recommendations. This method preserves the multifaceted nature of user interests, allowing the system to suggest content that reflects the diversity of topics or formats the user enjoys. In contrast, the averaging approach tended to overgeneralize, diluting nuanced preferences and increasing the risk of irrelevant recommendations, particularly when user interests span multiple domains.

In addition to technical design considerations, the recommendation engine is developed in full alignment with the evolving European regulatory framework. Specifically, the system respects the principles of the EU AI Act [24], ensuring transparency and accountability in algorithmic decision-making; it adheres to the requirements of the Digital Services Act (DSA) [25], which mandates responsible content curation and safeguards against systemic risks; it complies with the General Data Protection Regulation (GDPR) [26], protecting user privacy and ensuring lawful processing of personal data; finally, it and it aligns with the provisions of the EU Data Act [27], which promotes fair access to and sharing of data while ensuring data sovereignty and user control over personal and non-personal data.

5. Airflow Orchestration & Deployment

5.1. Overview

As outlined in previous sections, the platform is deployed in a multi-server architecture designed for scalability, resilience, and efficient workload orchestration. This design supports horizontal scaling, enabling components to expand independently, and improves fault tolerance by isolating failures to ensure service continuity. The deployment includes two Web Servers for content

 doi:10.20944/preprints202511.0449.v1

delivery and user interactions, and a dedicated AI Server optimized for GPU-intensive tasks such as embedding generation, semantic search, and real-time recommendations. By decoupling computationally expensive AI processes from request-serving operations, the system avoids bottlenecks and improves responsiveness. Apache Airflow serves as the core orchestration framework, coordinating ingestion, synchronization, and model updates across servers. Airflow manages complex pipelines in a reproducible, modular, and fault-tolerant manner. Its scheduling automates recurring tasks, while monitoring and retries mitigate transient failures. For our use case, Airflow offers three main advantages: seamless integration with relational and vector databases, allowing efficient ETL of podcast and user metadata into Qdrant; support for incremental upserts, keeping content and interaction data fresh without full re-indexing; and a DAG-based abstraction that structures pipelines as modular, interdependent tasks, making the system extensible as new data sources or processing stages are introduced.

5.2. Podcast and User Pipelines

Our system uses Apache Airflow as its main orchestration framework to manage data flows, ensuring they are reproducible, transparent, and fault-tolerant [11,12]. Airflow's key benefits include its ability to:

- Coordinate Data Pipelines: It seamlessly integrates with both relational and vector databases, allowing for efficient ETL (extract, transform, load) processes for podcast and user data.
- Handle Data Updates: It supports incremental updates, which keeps both content and user interaction data fresh without needing to re-index everything.
- Provide a Modular Structure: Its DAG (Directed Acyclic Graph) framework allows for pipelines to be broken down into a series of modular, interdependent tasks, making it easy to add new data sources or processing stages.

The Airflow instance is deployed directly on the AI Server to give its DAGs efficient access to GPU resources, which are crucial for tasks like embedding generation and model updates. This setup ensures that the entire recommendation pipeline remains synchronized and up-to-date. Airflow orchestrates the system through two categories of DAGs: Podcast Pipelines and User Pipelines, each designed to manage distinct lifecycle operations while preserving consistency across metadata, embeddings, and recommendations.

- Podcast Pipelines address ingestion, updates, and deletions of content. Insert DAGs begin by retrieving pending podcast entries from the job queue, then apply audio classification (BEATs), generate transcripts (FasterWhisperXXL and VAD), refine them with Gemini-2.5 flash-lite, label content with MPNet, and generate embeddings with LaBSE. These embeddings are inserted into Qdrant, synchronized with external APIs, and jobs are marked as complete. Update DAGs follow a similar path but emphasize refreshing labels, embeddings, and user profiles linked to the modified podcast. Delete DAGs ensure that obsolete embeddings are removed from Qdrant and that affected user embeddings are updated accordingly. To balance timeliness with efficiency, podcast DAGs execute in 30-minute cycles, which ensures new uploads and updates are integrated quickly without overloading resources.
- User Pipelines govern how user data influences the recommendation system. Insert DAGs process new user accounts by generating embeddings from initial listening histories or engagement data. Update DAGs refresh metadata (e.g., user country flag) and adjust embeddings to align with recent activity. Delete DAGs handle account removals or the loss of specific attributes, recomputing embeddings so user profiles remain accurate and aligned with the current dataset. Because user actions (listening, liking, following/unfollowing) directly affect recommendation quality, these DAGs run in 5-minute cycles, allowing the system to adapt almost immediately to behavioral changes.

This structure decouples content ingestion from user activity, aligning orchestration frequency with the different dynamics of podcasts (batch-oriented, periodic) and users (sensitive to real-time

 doi:10.20944/preprints202511.0449.v1

interactions). Together, the pipelines sustain a recommendation engine that is both dynamic and computationally efficient and are summarized in Table4.

Table 4. DAG—Summarization of execution cycles.

Component	Configuration	Rationale	Key Features
Episode-related DAGs	30-minute execution cycles	Optimal balance between data freshness and resource efficiency	Batch-oriented processing, no real-time requirements, modular structure allows dynamic rescheduling
User-related DAGs	5-minute execution cycles	Higher sensitivity to real-time user interactions	Near real-time adaptation to behavioral changes, maintains recommendation quality

5.3. End-to-End Workflow

When a user uploads a new podcast through the Web UI, the platform initiates a sequence of backend operations that connect the Web Server, AI Server, and orchestration layer. The Web UI first validates the user's permissions and stores the uploaded file in the NAS object storage, along with metadata in the Web Server's database. Once the upload is complete, the backend issues a request to the AI Server's API containing the metadata and storage path. This request is recorded in the MySQL job queue, where it awaits execution by the appropriate Airflow DAG.

At the next scheduled cycle, Airflow retrieves the pending job and triggers the corresponding Podcast Pipeline DAG. The DAG coordinates transcription, correction, classification, topic modeling, embedding generation, and synchronization tasks, preparing the podcast for search and recommendation. Similarly, user-related interactions—such as listening to or liking a podcast—are recorded and processed by User Pipeline DAGs, ensuring that recommendation vectors reflect the most recent behavior.

Once processing is complete, embeddings and metadata are updated in Qdrant and synchronized with external APIs, making the new or modified content immediately available for semantic search and personalized recommendations. From the user's perspective, the transition is seamless: a newly uploaded podcast becomes discoverable within minutes, and user preferences dynamically influence recommendations almost in real time. For detailed breakdowns of individual DAG tasks, readers are referred to Appendix C.

6. Implementation Considerations

The system architecture is deliberately designed for scalability, modularity and maintainability, employing a multi-server configuration: two dedicated Web Servers for user-facing services and separate AI Server for computationally intensive inference workloads. This separation of concerns facilitates efficient load balancing, fault tolerance, and the ability to scale each component independently. As the content library grows and user demand increases, this architecture enables seamless horizontal scaling of AI resources without affecting the responsiveness of the user interface.

Inter-server communication is implemented through RESTful and asynchronous APIs, supporting real-time interaction while maintaining decoupled processing. The design accommodates both batch-mode ingestion and offline indexing for resource-intensive operations—such as transcription, embedding generation, and model retraining—while preserving low-latency responses for end users. Incremental indexing and event-driven synchronization between the AI and Web layers ensure timely updates to search results and recommendations without requiring full reprocessing of the dataset.

 doi:10.20944/preprints202511.0449.v1

Long-term recommendation quality and retrieval accuracy are maintained through a robust machine learning lifecycle management strategy. This includes periodic model retraining, automated pipeline scheduling via Apache Airflow, and continuous monitoring for data and concept drift. The system leverages feedback signals—such as changes in user engagement, evolving content trends, and variations in audio characteristics—to proactively adapt predictive models. This approach mitigates performance degradation over time and supports the seamless integration of new models or functionalities, including multilingual understanding, topic adaptation, and advanced personalization algorithms.

The platform leverages over 350GB of audio content from more than thirteen years of European School Radio (ESR) archival material. This corpus includes podcasts, user-generated metadata, and listening statistics. Data processing complies with the GDPR [26] and the EU Data Act [27], ensuring fair access and responsible use of the available data. Metadata is curated using standards like Dublin Core [28] and www.schema.org to ensure interoperability. Long-term preservation is ensured through NAS infrastructure with RAID—5 and AES—256 encryption. Ethics oversight includes data minimization, user consent mechanisms, and transparency in AI operations.

7. Conclusions

Our podcast platform significantly elevates the user experience by offering intelligent, context-aware content discovery mechanisms tailored to both casual listeners and professional users such as educators. At the core of this discovery engine lies a combination of automated transcription, semantic search, and topic modeling, which collectively enable users to locate relevant audio content with unprecedented precision and speed. Rather than relying solely on superficial metadata like episode titles or manually entered keywords, the system understands the deeper semantic content of each episode. This allows users to uncover podcasts aligned with their interests—even when specific phrases or terminology are not explicitly mentioned.

Navigation is further enhanced through topic-based categorization, enabling exploration by broad thematic areas such as science, history, health, education, language learning, or social issues. These curated topical pathways are particularly valuable for teachers and educators, who can integrate podcast episodes into lesson plans, use them as supplementary materials to support classroom discussions, or assign them for homework and language practice. The availability of timestamped transcriptions also allows for precise referencing and the creation of educational worksheets or quizzes based on real spoken content.

Personalization features augment this discovery process by dynamically learning from each user's behavior—such as listening history, likes, or skipped episodes—and adapting recommendations in real time. This continual personalization not only improves satisfaction and long-term user retention but also encourages deeper engagement with the platform's growing content library.

Moreover, the platform supports discovery through its ability to reveal connections across different podcasts or series. For example, a listener exploring a history episode might recommend a linguistics podcast that touches on the evolution of historical narratives, thus bridging thematic gaps and enriching the listening journey. This cross-domain recommendation capability transforms passive listening into a personalized, exploratory experience that is both intellectually stimulating and highly relevant.

Author Contributions: All authors have contributed to the review presented. Conceptualization, I.K, G.P, K.D, A.G; methodology, I.K, G.P, K.D, M.D; data curation, I.K and G.P.; writing—original draft preparation, I.K and G.P.; writing—review and editing, I.K, G.P., M.D. and E.T.; visualization, I.K and G.P.; supervision, K.D.; project administration, K.D and A.G. All authors have read and agreed to the published version of the manuscript.

 doi:10.20944/preprints202511.0449.v1

Acknowledgments: The authors of this paper would like to thank the following individuals for their advice and support: From the Fédération nationale des Francas: Hervé Prévost, Myrto Stamelaki, From the Technische Jugendfreizeit- und Bildungsgesellschaft (tjfbg) gGmbH: Susanne Böhmig, Harald Schmidt.

Funding: This study is part of the European project "Children Radio Europe" (Project GAP 101136199 / CREA-CROSS-2023-MEDIA LITERACY), funded by the European Union under the Creative Europe program.

Data Availability Statement: The data used in this study consists of user-generated podcast recordings and metadata from the European School Radio and Youth Radio platforms. Due to privacy and ethical restrictions in compliance with the GDPR, these data cannot be publicly shared. All relevant system-level metric and evaluation results are included in this article. Additional information or access to restricted datasets may be provided by the authors upon reasonable request and subject to institutional data protection policies.

Conflicts of Interest: The authors declare no conflicts of interest.

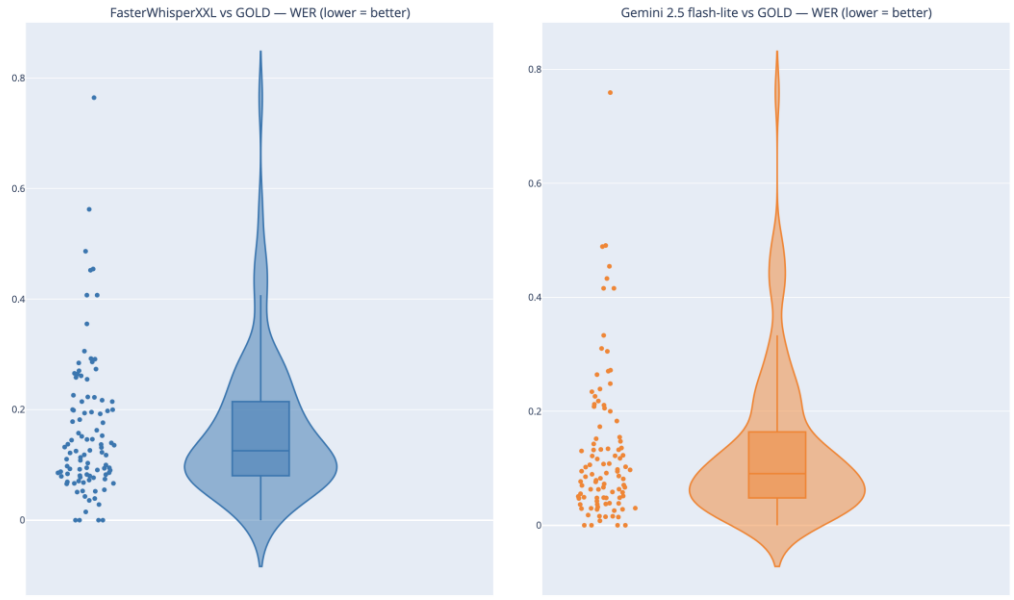
Abbreviations

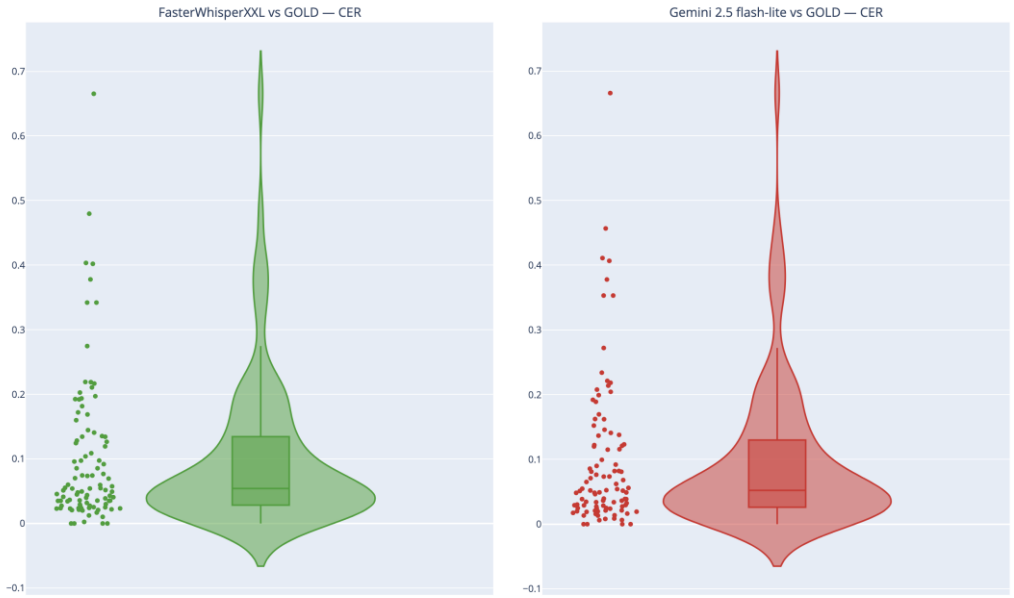
The following abbreviations are used in this manuscript:

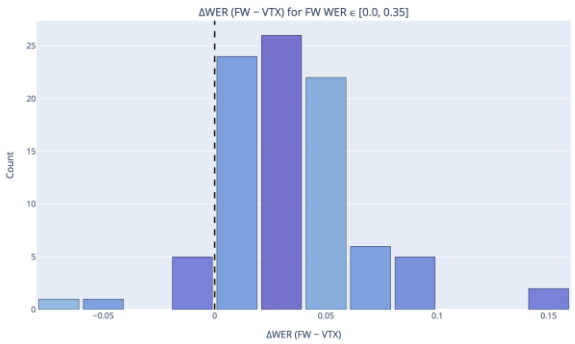
Acronym	Explanation
AI	Artificial Intelligence
ASR	Automatic Speech Recognition
API	Application Programming Interface
ANN	Approximate Nearest Neighbor (search)
BEATs	Bootstrap your Own Audio Transformer with Self-supervised Learning
BERTopic	Bidirectional Encoder Representations Topic Modeling (framework based on transformers)
BART	Bidirectional and Auto-Regressive Transformers
CER	Character Error Rate
CRUD	Create, Read, Update, Delete (basic database operations)
CSV	Comma-Separated Values
CTF-IDF	Class-based Term Frequency–Inverse Document Frequency
DAG	Directed Acyclic Graph (used in Airflow for workflow orchestration)
DCMI	Dublin Core Metadata Initiative
DSA	Digital Services Act
ESR	European School Radio
ETL	Extract, Transform, Load (data pipeline process)
EU AI Act	European Union Artificial Intelligence Act
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HNSW	Hierarchical Navigable Small World (graph-based ANN index)
IR	Information Retrieval
LaBSE	Language-agnostic BERT Sentence Embedding
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine Learning
NAS	Network Attached Storage
NDCG	Normalized Discounted Cumulative Gain (ranking metric)
PUA	Private Use Area (Unicode codepoints)
Qdrant	Open-source Vector Database for semantic search
RAID	Redundant Array of Independent Disks (RAID-5 specifically used here)
REST	Representational State Transfer (API style)
TF-IDF	Term Frequency–Inverse Document Frequency
TREC	Text REtrieval Conference (Podcast Track)
UI	User Interface
VAD	Voice Activity Detection
WER	Word Error Rate

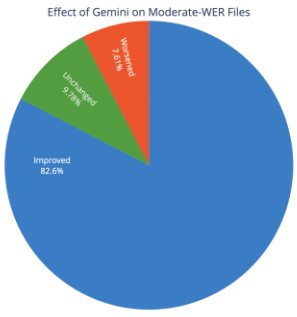
 doi:10.20944/preprints202511.0449.v1

Appendix A: Model Performance Metrics









Appendix B: List of Predefined Labels Used for Classification

- Humor

 doi:10.20944/preprints202511.0449.v1

- School Dialogue, Expressions
- Cinema and Directing
- Bullying, School Safety
- Theatre and Performing Arts
- European Elections and Democracy
- Charity, Volunteering
- Christmas and Holidays
- Sports
- Castles, Fortresses, Empires
- Video Watching
- Environment, Climate, Sustainability
- Radio and Broadcasts
- Road Safety, Accidents
- Gardening and Outdoor Spaces
- Artificial Intelligence, Technology
- Dostoevsky, Greek Perspectives
- Friendship and Social Bonds
- Disability, Education, Inclusion
- Nutrition and Health
- Books and Authors
- Football, Championships, Goals
- Poetry
- Travel and Routes
- Music
- Cars and Vehicles
- Announcements
- Pets and Animals
- School Life
- Spanish Language
- Video Games, Gaming
- Women Stereotypes
- Multilingual Excerpts
- Peace and Non-Violence
- Memories and Nostalgia
- Stories, Narration, Events
- German Culture
- Mothers, Family, Parenting
- French Culture
- History
- Internet and Digital World
- Puberty
- Fashion and Style
- Composers, Singers, Performers
- Cities, Countries, Refugees
- Health, Community, COVID
- Fairy Tales and Food

Appendix C: DAG

Podcast Pipelines

Podcast Insert DAG

- Read pending jobs from MySQL queue.

 doi:10.20944/preprints202511.0449.v1

- Apply BEATs for audio classification.
- Generate transcript with FasterWhisperXXL (large-v2-ct2 + VAD).
- Refine transcript with Gemini-2.5-Flash (Vertex AI).
- Apply BART model for label classification.
- Generate embeddings with LaBSE (title, description, transcript).
- Upsert embeddings into Qdrant.
- Synchronize outputs with external APIs (Youth Radio / European School Radio).
- Mark jobs as executed.

Podcast Update (Without Audio File) DAG

- Read pending jobs from MySQL queue.
- Apply BART model for label classification.
- Apply BEATs for audio classification.
- Generate embeddings for title, description, transcript.
- Upsert embeddings into Qdrant.
- Synchronize outputs with external APIs.
- Mark jobs as executed.
- Update user embeddings for listeners of the updated podcast.

Podcast Delete DAG

- Read pending jobs from MySQL queue.
- Delete embeddings from Qdrant.
- Mark jobs as executed.
- Update user embeddings for affected listeners.

User Pipelines

- User Insert DAG
- Read pending jobs from database.
- Perform integrity check (ensure episode embeddings exist).
- Generate user embeddings from listening history or engagement.
- Finalize database updates and mark jobs as completed.

User Update DAG

- Read pending jobs from database.
- Update user country flag in Qdrant metadata.
- Commit updates to database.

User Delete DAG

- Read pending jobs from database.
- Recompute embeddings to reflect attribute removals.
- Finalize updates and mark jobs as completed.

References

1. Jones, R.; Zamani, H.; Schedl, M.; Chen, C.-W.; Reddy, S.; Clifton, A.; Karlgren, J.; Hashemi, H.; Pappu, A.; Nazari, Z.; Yang, L.; Semerci, O.; Bouchard, H.; Carterette, B. *Current Challenges and Future Directions in Podcast Information Access. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*, Virtual Event, 11–15 July 2021; ACM: New York, NY, USA, 2021; pp. 1554–1565. Available online: https://dl.acm.org/doi/10.1145/3404835.3462805 (Accessed on 4 October 2025).
2. Jones, R.; Carterette, B.; Clifton, A.; Eskevich, M.; Jones, G. J. F.; Karlgren, J.; Pappu, A.; Reddy, S.; Yu, Y. *TREC 2020 Podcasts Track Overview,* arXiv, 2021. Available online: https://arxiv.org/abs/2103.15953 (Accessed on 4 October 2025)
3. Ghinassi, I.; Wang, L.; Newell, C.; Purver, M. *Multimodal Topic Segmentation of Podcast Shows with Pre-trained Neural Encoders. In Proceedings of the 2023 ACM International Conference on Multimedia Retrieval (ICMR '23),* Thessaloniki, Greece, 12–15 June 2023; Association for Computing Machinery: New York, NY, USA, 2023;

 doi:10.20944/preprints202511.0449.v1

pp. 602–606. Available online: https://dl.acm.org/doi/10.1145/3591106.3592270 (Accessed on 4 October 2025)

4. Manakul, P.; Yang, M.; Gales, M. *CUED_speech at TREC 2020 Podcast Summarisation Track. arXiv* 2020, arXiv:2012.02535. Available online: https://arxiv.org/abs/2012.02535 (Accessed on 4 October 2025).
5. Clifton, A.; Févotte, C.; Jones, R.; King, S.; Max, A.; Watanabe, S.; Zhang, Y.; Bell, P.; Gales, M.; Garimella, V.R.K.; et al. *100,000 Podcasts: A Spoken English Document Corpus. In Proceedings of the 28th International Conference on Computational Linguistics (COLING 2020),* Barcelona, Spain (Online), 8–13 December 2020; International Committee on Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 5903–5917. Available online: https://aclanthology.org/2020.coling-main.519 (Accessed on 4 October 2025).
6. Clifton, A.; Févotte, C.; Jones, R.; King, S.; Max, A.; Watanabe, S.; Zhang, Y.; Bell, P.; Gales, M.; Garimella, V.R.K.; et al. *The Spotify Podcast Dataset. arXiv* 2020, arXiv:2004.04270. Available online: https://arxiv.org/abs/2004.04270 (Accessed on 4 October 2025).
7. Alexander, A.; Jones, R.; Eskevich, M.; Ferrom, S.; Pecina, P.; Jones, G.J.F. *Audio Features, Precomputed for Podcast Retrieval and Information Access Experiments. In Experimental IR Meets Multilinguality, Multimodality, and Interaction (CLEF 2021)*; Springer: Cham, Switzerland, 2021; Lecture Notes in Computer Science, Vol. 12880; pp. 3–14. Available online: https://doi.org/10.1007/978-3-030-85251-1_1 (Accessed on 4 October 2025).
8. Ghazimatin, A.; Garmash, E.; Penha, G.; Sheets, K.; Achenbach, M.; Semerci, O.; Galvez, R.; Tannenberg, M.; Mantravadi, S.; Narayanan, D. et al. *PODTILE: Facilitating Podcast Episode Browsing with Auto-Generated Chapters. In Proceedings of the 2024 ACM International Conference on Information and Knowledge Management (CIKM '24).* Available online: https://arxiv.org/abs/2410.16148 (Accessed on 4 October 2025).
9. Meggetto, F.; Moshfeghi, Y. *Podify: A Podcast Streaming Platform with Automatic Logging of User Behaviour for Academic Research. In Proceedings of the 2023 ACM SIGIR Conference (SIGIR '23 — Demos),* 2023. Available online: https://dl.acm.org/doi/10.1145/3539618.3591824 (Accessed on 4 October 2025).
10. Paraskevopoulos, G.; Tsoukala, C.; Katsamanis, A.; Katsouros, V. *The Greek Podcast Corpus: Competitive Speech Models for Low-Resourced Languages with Weakly Supervised Data. In Proceedings of Interspeech 2024*, pp. 3969–3973. Available online: https://arxiv.org/pdf/2406.15284v1 (Accessed on 4 October 2025).
11. Kotliar, M.; Kartashov, A.V.; Barski, A. *CWL-Airflow: A Lightweight Pipeline Manager Supporting Common Workflow Language. GigaScience* 2019, 8(7), giz084. Available online: https://academic.oup.com/gigascience/article/8/7/giz084/5535758 (Accessed on 4 October 2025).
12. Yasmin, J.; Wang, J.; Tian, Y.; Adams, B. *An Empirical Study of Developers' Challenges in Implementing Workflows as Code: A Case Study on Apache Airflow. arXiv* 2024, arXiv:2406.00180. Available online: https://arxiv.org/abs/2406.00180 (Accessed on 4 October 2025).
13. Ahmed, A. E.; Allen, J. M.; Bhat, T.; Burra, P.; Fliege, C. E.; Hart, S. N.; Heldenbrand, J. R.; Hudson, M. E.; Istanto, D. D.; Kalmbach, M. T., et al. *Design Considerations for Workflow Management Systems in production genomics research and the clinic. Scientific Reports* 2021, 11, 21680. Available online: https://www.nature.com/articles/s41598-021-99288-8 (Accessed on 4 October 2025).
14. Purfview. *whisper-standalone-win: Whisper & Faster-Whisper Standalone Executables*. GitHub repository, 2024. Available online: https://github.com/Purfview/whisper-standalone-win (Accessed on 10 August 2025).
15. Google DeepMind, "Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities," Available online: https://arxiv.org/html/2507.06261v1 (Accessed on 10 August 2025).
16. Chen, S.; Wu, Y.; Wang, C.; Liu, S.; Tompkins, D.; Chen, Z.; Che, W.; Yu, X.; Wei, F. *BEATs: Audio Pre-Training with Acoustic Tokenizers. In Proceedings of the 40th International Conference on Machine Learning (ICML 2023);* Proceedings of Machine Learning Research; Volume 202; PMLR: Honolulu, HI, USA, 23–29 July 2023; pp. 5178–5193. Available online: https://proceedings.mlr.press/v202/chen23ag/chen23ag.pdf (Accessed on 4 October 2025).
17. Grootendorst, M. *BERTopic: Neural Topic Modeling with a Class-Based TF–IDF Procedure. arXiv* 2022, arXiv:2203.05794. Available online: https://arxiv.org/abs/2203.05794 (Accessed on 4 October 2025).
18. Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020). MPNet: Masked and Permuted Pre-training for Language Understanding. Available online: https://arxiv.org/abs/2004.09297 (Accessed on 4 October 2025).

 doi:10.20944/preprints202511.0449.v1

19. Feng, F.; Yang, Y.; Cer, D.; Arivazhagan, N.; Wang, W. *Language-Agnostic BERT Sentence Embedding. arXiv* 2020, arXiv:2007.01852. Available online: https://arxiv.org/abs/2007.01852 (Accessed on 4 October 2025).
20. Radford, A.; Kim, J.W.; Xu, T.; Brockman, G.; McLeavey, C.; Sutskever, I. *Robust Speech Recognition via Large-Scale Weak Supervision. arXiv* 2022, arXiv:2212.04356. Available online: https://arxiv.org/abs/2212.04356 (Accessed on 4 October 2025).
21. Ma, R.; Qian, M.; Gales, M.; Knill, K. *ASR Error Correction Using Large Language Models. arXiv* 2024, arXiv:2409.09554. Available online: https://arxiv.org/abs/2409.09554 (Accessed on 4 October 2025).
22. Szymański, P.; Żelasko, P. *WER We Are and WER We Think We Are: Skepticism on Very Low ASR Error Rates. Findings of the Association for Computational Linguistics: EMNLP 2020*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 455–467. Available online: https://aclanthology.org/2020.findings-emnlp.295.pdf (Accessed on 4 October 2025).
23. D. K. Thennal et al., "Advocating Character Error Rate for Multilingual ASR Evaluation," *Findings of NAACL*, 2025. [Online]. Available: https://aclanthology.org/2025.findings-naacl.277.pdf
24. European Union. *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act) and Amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2016/798*; Official Journal of the European Union, L 202, pp. 1–180, 12 July 2024. Available online: https://eur-lex.europa.eu/eli/reg/2024/1689/oj (Accessed on 4 October 2025).
25. European Union. *Regulation (EU) 2022/2065 of the European Parliament and of the Council of 19 October 2022 on a Single Market for Digital Services and Amending Directive 2000/31/EC (Digital Services Act);* Official Journal of the European Union, L 277, pp. 1–102, 27 October 2022. Available online: https://eur-lex.europa.eu/eli/reg/2022/2065/oj (Accessed on 4 October 2025).
26. European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data (General Data Protection Regulation);* Official Journal of the European Union, L 119, pp. 1–88, 4 May 2016. Available online: https://eur-lex.europa.eu/eli/reg/2016/679/oj (Accessed on 4 October 2025).
27. *European Parliament and Council of the European Union, Regulation (EU) 2023/2854 of the European Parliament and of the Council of 13 December 2023 on harmonised rules on fair access to and use of data (Data Act), Official Journal of the European Union, L 2023/2854, Dec. 22, 2023.*
28. International Organization for Standardization (ISO). *ISO 15836-1:2017 — Information and Documentation — The Dublin Core Metadata Element Set — Part 1: Core Elements;* ISO: Geneva, Switzerland, 2017. Available online: https://www.iso.org/standard/71339.html (Accessed on 4 October 2025).