Architectural Blueprint for High-Fidelity Hierarchical Audio Transcription Systems: Integrating Faster-Whisper, Qwen2-Audio, and Next.js

1. Executive Summary and Architectural Philosophy

The rapid evolution of Automatic Speech Recognition (ASR) has transitioned from Hidden Markov Models to end-to-end deep learning architectures, culminating in systems like OpenAI's Whisper. While these models achieve remarkable Word Error Rates (WER), they function primarily as acoustic-to-text transducers, lacking the semantic reasoning required to self-correct during ambiguous acoustic events. This report outlines the comprehensive design, implementation, and operational strategy for a **Confidence-Aware Hierarchical Transcription System (CAHTS)**. This architecture addresses the stochastic limitations of standard ASR by implementing a secondary "cognitive" review layer using Qwen2-Audio, a multimodal Large Language Model (LLM), orchestrated within a robust Next.js and Docker environment.

The core engineering challenge addressed here is the orchestration of heterogeneous inference engines—a lightweight, high-throughput ASR (Faster-Whisper) and a heavy, semantic-reasoning multimodal model (Qwen2-Audio)—without compromising user experience or system stability. By leveraging a "verify-and-correct" loop, the system identifies phonemically unstable segments (confidence < 60%) and surgically re-processes them. This hierarchical approach optimizes computational resources, reserving expensive multimodal inference only for data segments that necessitate semantic disambiguation.

This document serves as a definitive technical reference for systems architects and full-stack engineers. It details the integration of a Next.js application with a custom SQLite-based job queue, Nginx Proxy Manager for secure ingress, and a complex signal processing pipeline involving FFmpeg and disparate AI endpoints. We employ a Test-Driven Development (TDD) methodology to ensure reliability and utilize Discord webhooks for event-driven observability.

2. Infrastructure and Orchestration: The Docker Ecosystem

The foundation of the CAHTS architecture is a containerized environment managed via Docker Compose. This choice facilitates reproducible deployments and network isolation while allowing controlled access to host-level resources. The infrastructure is composed of three primary service vectors: the Application Plane (Next.js), the Gateway Plane (Nginx Proxy Manager), and the Inference Plane (External Whisper and Containerized Qwen2-Audio).

2.1 Network Topology and Service Discovery

A critical requirement of this architecture is the hybrid nature of the inference resources. The user specifies an *existing* external faster-whisper server running on the host machine (port 62277), while the Qwen2-Audio service may be containerized or accessed remotely. This necessitates a split-horizon networking strategy.

Docker containers, by default, run in an isolated bridge network. To access a service running on the host machine's loopback interface from within a container, one cannot simply use localhost. The architecture must employ the special DNS mapping host.docker.internal. On Linux systems, this mapping is not automatic and requires the extra_hosts directive in the docker-compose.yml file, mapping the hostname to the host-gateway.[1] This configuration allows the Next.js application to "break out" of its containerized environment to query the high-performance Whisper backend running natively on the host hardware—a crucial optimization to leverage native CUDA drivers without complex container GPU passthrough for the primary ASR.

Conversely, the internal services—Next.js, Nginx Proxy Manager (NPM), and the Qwen2-Audio server (if running via vLLM)—communicate over a dedicated user-defined bridge network, termed here as transcription_net. This isolation ensures that internal traffic, such as database polling and unencrypted inference requests between the app and Qwen2, remains unexposed to the host or public interfaces.[3]

2.2 Ingress Management with Nginx Proxy Manager

Nginx Proxy Manager (NPM) serves as the architectural gateway. While a raw Nginx configuration could suffice, NPM provides a GUI-driven abstraction layer that simplifies the management of SSL certificates (via Let's Encrypt) and complex routing rules ("Proxy Hosts"). In a production-grade homelab or enterprise intranet, separating the SSL termination from the application logic is a security best practice.[5]

The docker-compose configuration for NPM requires mapping ports 80, 443, and 81 (management UI) to the host. A persistent volume strategy is essential here; NPM relies on a SQLite database (by default) or a MariaDB backend to store configuration state. For this architecture, mapping ./data and ./letsencrypt ensures that SSL certificates and routing rules persist across container restarts.[6]

Table 1: Docker Service Configuration and Resource Allocation

Service Component	Container Strategy	Network Context	Critical Configuration
Ingress Gateway	jc21/nginx-proxy-manager	Public Bridge + Internal	network_mode: bridge for ingress; connects to transcription_net for upstream.
Application Core	Node.js (Alpine)	transcription_net	extra_hosts: - "host.docker.internal:host-gateway" to access Whisper.
Qwen2 Inference	vllm/vllm-openai	transcription_net	--shm-size=16gb for tensor parallelism; GPU passthrough enabled.
State Store	SQLite (File-based)	N/A (Volume Mount)	Shared volume between App and Worker services.

2.3 Qwen2-Audio Containerization Challenges

Deploying the secondary inference engine, Qwen2-Audio, presents specific challenges regarding memory management. Large Multimodal Models (LMMs) like Qwen2-Audio-7B require significant Video RAM (VRAM). Standard deployments using vllm or ollama often fail inside Docker if the shared memory size (/dev/shm) is insufficient, as PyTorch and NCCL libraries utilize shared memory for inter-process communication during tensor parallel operations.[8]
The Docker Compose definition for the Qwen2 service must explicitly set shm_size: '16gb' (or approx. 8GB minimum for quantized models) and enable the deploy.resources.reservations.devices capability to pass the NVIDIA GPU into the container. Without this, the inference server will either fail to start or fallback to CPU, rendering the 20-second correction processing prohibitively slow.[8] We utilize vllm as the serving engine due to its high throughput and native support for OpenAI-compatible APIs, which simplifies the integration code within the Next.js application.[10]

3. The Ingestion Layer: Next.js and Asynchronous Patterns

Next.js is selected for the application layer due to its hybrid capability to serve React frontends and execute server-side API logic. However, a fundamental constraint of modern web frameworks (and serverless-like deployments) is the "Execution Timeout." Transcribing audio, even with faster-whisper, is a long-running process that often exceeds the default 30-60 second timeout of Nginx or Vercel-like environments.[12]

3.1 The Decoupled Job Architecture

To circumvent timeout issues, CAHTS implements a strict **Asynchronous Job Pattern**. The architecture rejects synchronous processing (POST /transcribe -> wait -> return text). Instead, it adopts a command-query separation (CQRS) approach:

1. **Submission (Command):** The client uploads the audio file via POST /api/upload. The server stores the file on a persistent volume, creates a Job record in SQLite with a status of PENDING, and immediately returns a 202 Accepted response containing the jobId.
2. **Processing (Worker):** A separate, long-lived process (the Worker) monitors the database. It claims the job, performs the transcription, FFmpeg slicing, and Qwen2 correction, and finally updates the database record.
3. **Retrieval (Query):** The client polls GET /api/jobs/[id] to check the status. Once COMPLETED, the client fetches the result.[14]

3.2 Implementation of the Polling Mechanism

While WebSockets or Server-Sent Events (SSE) offer real-time push capabilities, they introduce significant complexity regarding connection management, especially when sitting behind a reverse proxy like Nginx Proxy Manager which may drop idle connections. For this scale of application, **Short Polling** (e.g., every 3-5 seconds) is the most robust and implementation-friendly approach. It is stateless, resilient to network interruptions, and easily cacheable.[14]

The frontend, built with React, utilizes a hook library such as SWR (Stale-While-Revalidate) or TanStack Query (React Query) to manage this polling. React Query is particularly well-suited here as it allows for "smart" polling—pausing when the window is out of focus or employing exponential backoff.[15] The hook queries the job status endpoint; if the status is PROCESSING, the hook continues to poll. If COMPLETED or FAILED, polling stops.

Deep Insight: The choice between SWR and React Query often comes down to ecosystem preference, but React Query's devtools and granular control over retry logic make it slightly superior for debugging complex asynchronous flows where a job might fail halfway through due to an FFmpeg error.[17]

4. Persistence and State Management: The SQLite

Queue

Using a database as a job queue is often criticized in high-throughput distributed systems, where specialized brokers like Redis (BullMQ) or RabbitMQ are preferred. However, for a single-node or small-cluster architecture like CAHTS, incorporating Redis adds unnecessary operational overhead (another container to manage, potential network latency). SQLite, particularly when configured in Write-Ahead Logging (WAL) mode, is more than capable of handling the concurrency required for this application.[18]

4.1 Schema Design and Type Safety

We employ Prisma ORM to define the schema. This choice provides strict type safety across the full stack—ensuring that the "Job Status" enums used in the database match those expected by the frontend and the worker script.[20]

The data model must support the hierarchical nature of the transcription. We need to store the master transcript, but also the granular analysis of the confidence scores to allow for auditing the AI's performance.

Table 2: Prisma Schema Definition for Hierarchical Transcription

Model Entity	**Field**	**Type**	**Purpose**
Job	id	String (UUID)	Unique identifier for the polling loop.
	status	Enum	PENDING, PROCESSING, COMPLETED, FAILED.
	originalAudioPath	String	File system path to the uploaded WAV/MP3.
	transcript	String (Text)	The final merged text output.
Segment	jobId	Relation	Links segments to the master job.
	start	Float	Start time in seconds (from Whisper).
	end	Float	End time in seconds.
	confidence	Float	Average log-probability converted to linear probability.
Correction	segmentId	Relation	Links a correction to a specific time segment.

	originalText	String	The text Whisper hallucinated or missed.
	correctedText	String	The text generated by Qwen2-Audio.
	triggerConfidence	Float	The confidence score that triggered this correction.

4.2 The "SKIP LOCKED" Pattern in SQLite

A critical challenge in database-backed queues is preventing multiple workers (if scaled horizontally) from claiming the same job. PostgreSQL offers FOR UPDATE SKIP LOCKED. SQLite does not natively support this syntax.
To emulate this safety in SQLite:

1. **WAL Mode:** Ensure PRAGMA journal_mode = WAL; is executed on connection. This allows simultaneous readers and writers.[19]
2. **Atomic Status Updates:** The worker does not simply SELECT and then UPDATE. Instead, it performs an atomic update or utilizes a reserved state.
   - *Algorithm:* The worker initiates a transaction. It selects the oldest PENDING job. If one exists, it immediately updates the status to PROCESSING and commits. If the commit succeeds, the worker owns the job. If the database is locked (SQLITE_BUSY), the worker backs off and retries.[22]

This mechanism ensures that even if the Next.js container spins up multiple worker threads, no two threads process the same audio file, preventing race conditions and file access conflicts.

5. Primary ASR: Faster-Whisper Integration and Confidence Analysis

The ingestion phase relies on the external faster-whisper server. This server is an OpenAI-compatible implementation of the CTranslate2-backed Whisper model, known for being up to 4x faster than the original PyTorch implementation.[1]

5.1 API Payload Configuration for Confidence Extraction

Standard usage of the OpenAI Transcription API returns a simple JSON object with a text

string. This is insufficient for our needs. To identify "words with <60% confidence," we must extract the granular token-level probabilities.
The request to http://host.docker.internal:62277/v1/audio/transcriptions must include:

1. response_format="verbose_json": This instructs the server to return a detailed object including segments and metadata.[25]
2. timestamp_granularities=["word"]: This is the crucial parameter. Without this, Whisper might only return segment-level averages. We need word-level precision to pinpoint exactly *which* word is uncertain.[25]

API Response Structure Analysis:
The verbose_json response contains a words array. Each item in this array looks like:

JSON

```json
{
  "word": "supercalifragilistic",
  "start": 12.4,
  "end": 13.1,
  "probability": 0.55
}
```

The probability field corresponds to the model's confidence (0.0 to 1.0). The architecture defines a constant CONFIDENCE_THRESHOLD = 0.60.

5.2 Clustering Algorithm for Efficiency

A naive implementation would iterate through every word, check if probability < 0.6, and immediately trigger a Qwen2 correction. This is inefficient. If a sentence is mumbled, we might have 10 sequential words with low confidence. Triggering 10 separate 20-second corrections (which would overlap significantly) wastes GPU cycles and creates merging chaos.
The Clustering Logic:

1. **Scan:** Iterate through the words array. Flag indices where probability < 0.6.
2. **Cluster:** Group consecutive or proximal flagged indices. If Word A (10.5s) and Word B (12.0s) are both low confidence, they belong to the same "ambiguity event." A proximity threshold (e.g., 5 seconds) determines the cluster boundaries.
3. **Windowing:** For each cluster, determine the center_time.
4. **Clip Definition:** Define the clip processing window as [center_time - 10s, center_time + 10s]. This ensures the Qwen2 model has 20 seconds of context to resolve the ambiguity.
5. **Deduplication:** Merge overlapping windows. If Cluster 1 requires 10s-30s and Cluster 2 requires 25s-45s, the system merges them into a single 10s-45s clip (or processes them

as a single block) to minimize FFmpeg operations.[1]

6. Signal Processing: FFmpeg and Audio Segmentation Strategy

Once the "ambiguity windows" are identified, the system must physically slice the audio file. This is handled by FFmpeg. The precision of this operation is paramount; if the slice is off by a few hundred milliseconds, the alignment between the new Qwen2 transcript and the original Whisper transcript will fail.

6.1 Precision Seeking vs. Stream Copying

FFmpeg offers two seeking modes:

- **Input Seeking (-ss before -i):** Fast, jumps to the nearest keyframe. For audio, this can be imprecise.[29]
- **Output Seeking (-ss after -i):** Decodes the stream up to the seek point. Highly accurate but slower.

Since our clips are short (20s) and the source files might be long, a hybrid approach or strictly accurate seeking is required. Furthermore, we cannot use -c copy (Stream Copy). Stream copying only cuts at audio frame boundaries, which might not align with the exact millisecond timestamp returned by Whisper.

The Re-Encoding Mandate:

We must re-encode the clip to a format optimized for Qwen2-Audio. Qwen2 typically expects raw waveform data or standard WAV files at a specific sampling rate (often 16kHz to match its pre-training).31

The Optimized FFmpeg Command:

Bash

```
ffmpeg -y -i input_master.mp3 -ss 124.50 -t 20.00 \
  -ac 1 -ar 16000 -c:a pcm_s16le \
  output_clip_124.wav
```

- -y: Overwrite without asking.
- -ss 124.50: Start time derived from the confidence cluster center.
- -t 20.00: Duration of the clip.
- -ac 1: Mix down to mono (LLMs generally process mono).
- -ar 16000: Resample to 16,000 Hz. This is critical. If the audio is 44.1kHz and Qwen

expects 16kHz, doing this in FFmpeg prevents the Python inference script from having to do it, saving inference time.[32]
- -c:a pcm_s16le: Encode as uncompressed WAV (signed 16-bit little-endian). This removes compression artifacts that could confuse the LLM.

7. Secondary Inference: Qwen2-Audio Configuration and Prompt Engineering

The correction phase utilizes Qwen2-Audio, a specialized Multi-Modal Large Language Model (MLLM) capable of understanding both audio and text instructions. This capabilities distinguishes it from Whisper; we can *ask* Qwen to "Transcribe this carefully" or "Identify the background noise."

7.1 Serving Infrastructure: vLLM vs. Ollama vs. LM Studio

The research indicates several ways to serve Qwen2-Audio:
1. **LM Studio:** Provides a GUI and an OpenAI-compatible local server. Good for testing, but hard to dockerize efficiently for a headless backend.[34]
2. **Ollama:** Supports Qwen models but has limited/experimental support for *audio* input via the API currently. Snippets suggest it's possible but may require custom Go processors or bleeding-edge builds.[35]
3. **vLLM:** The recommended production path. It supports high-throughput serving, OpenAI compatibility, and explicit tensor parallelism. It can handle the audio input via multimodal API extensions.[8]

We select **vLLM** for the Docker stack. It allows us to define the model Qwen/Qwen2-Audio-7B-Instruct and expose an endpoint that our Next.js app can hit.

7.2 Constructing the Multimodal Payload

Unlike a text-only LLM, the payload to Qwen2-Audio must carry the audio signal. vLLM supports passing audio via a URL or a base64 encoded string. Since the Next.js container and the vLLM container share a network but not necessarily a file system (unless we set up a complex shared volume), sending the 20-second clip as a **base64 string** is the most robust architectural choice.[38]
The Request Payload:

JSON

```
{
  "model": "Qwen/Qwen2-Audio-7B-Instruct",
  "messages":
    }
  ]
}
```

7.3 Prompt Engineering for Transcription Integrity

Qwen2-Audio is chatty. If asked "What did he say?", it might reply, "The speaker said 'Hello'."
We need raw transcription.
The system prompt or user instruction must be rigorous: "Transcribe the speech in this audio clip exactly. Do not add preamble. Do not translate. If the audio is unintelligible, output."
This prompt engineering ensures that the output can be directly swapped into the master transcript without manual cleaning.40

8. The Reconciliation Engine: Merging and Alignment

Once Qwen2 returns the corrected text for the 20-second clip, the system must merge it back into the master Whisper transcript. This is the most algorithmic complex part of the pipeline.

8.1 The Alignment Problem

Whisper gave us text with timestamps: "The red fox."
Qwen gave us text without timestamps: "The quick red fox."
We know the Qwen clip covered 0:05 to 0:25.
Reconciliation Algorithm:

1. **Anchor Identification:** Identify "Anchor Words" at the edges of the Qwen transcript that match high-confidence words in the Whisper transcript.
    - *Example:* Whisper had "The red fox." Qwen has "The quick red fox."
    - "The" and "fox" are anchors.
2. **Replacement:** The text *between* the anchors in Whisper is replaced by the text *between* the anchors in Qwen.
3. **Timestamp Interpolation:** Since Qwen doesn't return timestamps, the system must interpolate the timestamps for the new words ("quick") based on the duration of the gap between the anchors.
    - If "The" ends at 0:10.5 and "red" starts at 0:11.0, and we insert "quick", we assign

"quick" the rough timestamp of 0:10.75.

If Qwen's output is drastically different (e.g., a completely different sentence structure), the system calculates the **Levenshtein Distance**. If the distance is too large (indicating a potential hallucination or a complete mismatch), the system discards the correction and reverts to the original Whisper text, flagging the segment as "Unresolved" in the database. This safety valve prevents the "correction" logic from degrading the transcript quality.[27]

9. Quality Assurance: Test-Driven Development with Vitest

Implementing such a complex pipeline without tests is inviting failure. We employ **Test-Driven Development (TDD)** using **Vitest**, which integrates natively with the Vite bundler used by Next.js.[43]

9.1 Mocking the External World

The primary challenge in testing this architecture is the reliance on external systems (Docker containers, Whisper API, Qwen API, File System). We cannot spin up the full Docker stack for every unit test.

Mocking Strategy:

1. **Service Mocks:** We create a WhisperService and QwenService interface. in the test environment, we use vi.mock to intercept calls to these services.
    - *Scenario:* "Should trigger correction." The mock Whisper service returns a JSON with a word probability of 0.4. The test asserts that the QwenService.transcribe method was called.
2. **FFmpeg Abstraction:** We wrap the child_process.exec or fluent-ffmpeg calls in an AudioProcessor class. In tests, we mock this class to verify that the command string generated contains the correct -ss and -t values based on the cluster logic, without actually requiring an MP3 file to exist.[44]
3. **Database Fixtures:** We use an in-memory SQLite database for the test runner. Prisma supports swapping the DATABASE_URL dynamically. This allows us to test the queue logic (PENDING -> PROCESSING -> COMPLETED) in isolation.[46]

Sample Test Specification:

- **Input:** A mock audio file path.
- **Mock Whisper:** Returns 3 words, middle word has 0.2 confidence.
- **Expectation:**
    1. Job status moves to PROCESSING.
    2. FFmpeg command is generated with correct timestamps.
    3. Qwen API is called with base64 data.

4. Final transcript contains the merged text.
5. Job status moves to COMPLETED.

10. Operational Awareness: Discord Notification Integration

System observability is crucial for background tasks. Users need to know when their file is ready without constantly refreshing the UI. Discord Webhooks provide a low-friction, push-notification system.

10.1 Payload Formatting and Embeds

Discord's webhook API expects a specific JSON structure. Simple text is boring; we utilize **Embeds** to provide a rich report.[48]
The Notification Payload:
The worker constructs a JSON object upon job completion:

JSON

```
{
  "username": "TranscriptionBot",
  "embeds":,
      "footer": {
        "text": "Processed by CAHTS v1.0"
      },
      "timestamp": "2023-10-27T10:00:00.000Z"
    }
  ]
}
```

This payload is sent via a standard POST request to the webhook URL stored in the .env file. The system handles rate limiting (HTTP 429) by respecting the Retry-After header returned by Discord.[50]

11. Conclusion and Strategic Outlook

The **Confidence-Aware Hierarchical Transcription System (CAHTS)** represents a

significant step forward in automated media processing. By decoupling the ingestion (Next.js) from the intelligence (Whisper/Qwen) using Docker, and bridging the gap between raw ASR and semantic understanding, the architecture delivers superior transcription fidelity.
This approach highlights a broader trend in AI engineering: the move from monolithic "do-it-all" models to **Chain-of-Model** pipelines. Whisper acts as the fast, efficient frontline, while Qwen2-Audio serves as the specialized expert called in only when necessary. This yields a cost-effective, high-accuracy system suitable for legal, medical, or archival domains where every word matters. The integration of TDD, SQLite queuing, and Discord notifications ensures that the system is not just a research prototype, but a robust, production-ready utility.

Citations:

1

Works cited

1. Faster Whisper transcription with CTranslate2 - GitHub, accessed November 18, 2025, https://github.com/SYSTRAN/faster-whisper
2. The Ultimate Guide to Running Nginx Proxy Manager via Docker - YouTube, accessed November 18, 2025, https://www.youtube.com/watch?v=kR01Is8Xa2M
3. How to Build and Run Next.js Applications with Docker, Compose, & NGINX, accessed November 18, 2025, https://www.docker.com/blog/how-to-build-and-run-next-js-applications-with-docker-compose-nginx/
4. Question: How do Docker networks work with Nginx Proxy Manager? - Reddit, accessed November 18, 2025, https://www.reddit.com/r/docker/comments/1eek641/question_how_do_docker_networks_work_with_nginx/
5. Nginx Proxy Manager Deployment | Blog of Hasitha Suneth, accessed November 18, 2025, https://blog.hasithasuneth.com/docs/deployments/nginx-proxy-manager/
6. NGINX Proxy Manager & Docker Install Guide - YouTube, accessed November 18, 2025, https://www.youtube.com/watch?v=LPQMCBQSTMw
7. Full Setup Instructions - Nginx Proxy Manager, accessed November 18, 2025, https://nginxproxymanager.com/setup/
8. Single NPU (Qwen2-Audio-7B) — vllm-ascend, accessed November 18, 2025, https://docs.vllm.ai/projects/ascend/en/main/tutorials/single_npu_qwen2_audio.html
9. Qwen 2.5 7B VRAM Tips Every Dev Should Know | by Novita AI - Medium, accessed November 18, 2025, https://medium.com/@marketing_novita.ai/qwen-2-5-7b-vram-tips-every-dev-should-know-932303373ff0
10. vLLM - Qwen, accessed November 18, 2025, https://qwen.readthedocs.io/en/latest/deployment/vllm.html

11. OpenAI-Compatible Server - vLLM, accessed November 18, 2025, https://docs.vllm.ai/en/v0.8.3/serving/openai_compatible_server.html
12. Creating a Scalable API for Long-Running FFmpeg Tasks : r/golang - Reddit, accessed November 18, 2025, https://www.reddit.com/r/golang/comments/1dojyvb/creating_a_scalable_api_for_longrunning_ffmpeg/
13. How do you handle long running tasks in Next? : r/nextjs - Reddit, accessed November 18, 2025, https://www.reddit.com/r/nextjs/comments/1fd3nz1/how_do_you_handle_long_running_tasks_in_next/
14. Long-Running Tasks with Next.js: A Story of Reinventing the Wheel - DEV Community, accessed November 18, 2025, https://dev.to/bardaq/long-running-tasks-with-nextjs-a-journey-of-reinventing-the-wheel-1cjg
15. useSWR vs. React Query – Differences and which one should you choose? - Codedamn, accessed November 18, 2025, https://codedamn.com/news/javascript/useswr-vs-react-query-differences-and-which-one-should-you-choose
16. Should I use React Query with NextJS when I'll only be pulling from my own routes and database? : r/reactjs - Reddit, accessed November 18, 2025, https://www.reddit.com/r/reactjs/comments/10251dn/should_i_use_react_query_with_nextjs_when_ill/
17. SWR vs React Query: The Ultimate Guide to Data Fetching in React Applications - Medium, accessed November 18, 2025, https://medium.com/@siddharthpatil9108/swr-vs-react-query-the-ultimate-guide-to-data-fetching-in-react-applications-7a8d6e5d737f
18. 'Don't use a database as a queue' : r/dotnet - Reddit, accessed November 18, 2025, https://www.reddit.com/r/dotnet/comments/1d8piwy/dont_use_a_database_as_a_queue/
19. A SQLite Background Job System - JasonGorman;, accessed November 18, 2025, https://jasongorman.uk/writing/sqlite-background-job-system/
20. How to use Docker Compose file with Next.js, Prisma, and SQLite - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/77783363/how-to-use-docker-compose-file-with-next-js-prisma-and-sqlite
21. Quickstart with TypeScript & SQLite | Prisma Documentation, accessed November 18, 2025, https://www.prisma.io/docs/getting-started/quickstart-sqlite
22. best practices for using sqlite for a database queue - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/1631051/best-practices-for-using-sqlite-for-a-database-queue
23. The best way to use a DB table as a job queue (a.k.a batch queue or message queue) - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/297280/the-best-way-to-use-a-db-table-as

-a-job-queue-a-k-a-batch-queue-or-message-queu
24. 5 Ways to Speed Up Whisper Transcription | Modal Blog, accessed November 18, 2025, https://modal.com/blog/faster-transcription
25. Speech to text - OpenAI API, accessed November 18, 2025, https://platform.openai.com/docs/guides/speech-to-text
26. etalab-ia/faster-whisper-server - GitHub, accessed November 18, 2025, https://github.com/etalab-ia/faster-whisper-server
27. How can I get word-level timestamps in OpenAI's Whisper ASR? - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/73822353/how-can-i-get-word-level-timestamps-in-openais-whisper-asr
28. Getting sentence level log probabilities / confidence scores · Issue #1358 · SYSTRAN/faster-whisper - GitHub, accessed November 18, 2025, https://github.com/SYSTRAN/faster-whisper/issues/1358
29. Cutting multimedia files based on start and end time using ffmpeg [closed] - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/18444194/cutting-multimedia-files-based-on-start-and-end-time-using-ffmpeg
30. Using ffmpeg to cut audio from/to position [closed] - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/46508055/using-ffmpeg-to-cut-audio-from-to-position
31. Qwen2-Audio Technical Report - arXiv, accessed November 18, 2025, https://arxiv.org/html/2407.10759v1
32. How to cut at exact frames using ffmpeg? - Super User, accessed November 18, 2025, https://superuser.com/questions/459313/how-to-cut-at-exact-frames-using-ffmpeg
33. Using ffmpeg to cut videos with more precision than key frames allow, accessed November 18, 2025, https://video.stackexchange.com/questions/16750/using-ffmpeg-to-cut-videos-with-more-precision-than-key-frames-allow
34. REST API v0 | LM Studio Docs, accessed November 18, 2025, https://lmstudio.ai/docs/developer/rest/endpoints
35. Feature Request: Add Audio Input Support for Multimodal Models · Issue #11798 - GitHub, accessed November 18, 2025, https://github.com/ollama/ollama/issues/11798
36. Ollama's new engine for multimodal models, accessed November 18, 2025, https://ollama.com/blog/multimodal-models
37. qwen2_audio - vLLM, accessed November 18, 2025, https://docs.vllm.ai/en/latest/api/vllm/model_executor/models/qwen2_audio/
38. Multimodal Inputs - vLLM, accessed November 18, 2025, https://docs.vllm.ai/en/v0.8.4/serving/multimodal_inputs.html
39. [Bug]: OpenAI-compatible server doesn't support audio inputs · Issue #19977 · vllm-project/vllm - GitHub, accessed November 18, 2025,

https://github.com/vllm-project/vllm/issues/19977
40. Qwen2-Audio: Chat with Your Voice! | Qwen, accessed November 18, 2025, https://qwenlm.github.io/blog/qwen2-audio/
41. Qwen/Qwen2-Audio-7B-Instruct - Hugging Face, accessed November 18, 2025, https://huggingface.co/Qwen/Qwen2-Audio-7B-Instruct
42. Qwen2Audio - Hugging Face, accessed November 18, 2025, https://huggingface.co/docs/transformers/en/model_doc/qwen2_audio
43. Testing: Vitest - Next.js, accessed November 18, 2025, https://nextjs.org/docs/pages/guides/testing/vitest
44. Mocking | Guide - Vitest, accessed November 18, 2025, https://vitest.dev/guide/mocking
45. How to test a Server Action using Testing Library? - nextjs - Reddit, accessed November 18, 2025, https://www.reddit.com/r/nextjs/comments/14lun3o/how_to_test_a_server_action_using_testing_library/
46. Usage with Worker Threads · Issue #237 · WiseLibs/better-sqlite3 - GitHub, accessed November 18, 2025, https://github.com/JoshuaWise/better-sqlite3/issues/237
47. Guided: Testing a Next.js Finance Application with Vitest - Pluralsight, accessed November 18, 2025, https://www.pluralsight.com/labs/codeLabs/guided-testing-a-nextjs-finance-application-with-vitest
48. Webhook Resource | Documentation | Discord Developer Portal, accessed November 18, 2025, https://discord.com/developers/docs/resources/webhook
49. Can't get discord webhook to send with json - Stack Overflow, accessed November 18, 2025, https://stackoverflow.com/questions/65906461/cant-get-discord-webhook-to-send-with-json
50. How to format Discord Webhook or how to use POST - CubeCoders Support, accessed November 18, 2025, https://discourse.cubecoders.com/t/how-to-format-discord-webhook-or-how-to-use-post/1572
51. Any guide to make generic webhook work with discord? · sct overseerr · Discussion #1074, accessed November 18, 2025, https://github.com/sct/overseerr/discussions/1074
52. Word level timestamps from whisper v3's json is invalid - OpenAI Developer Community, accessed November 18, 2025, https://community.openai.com/t/word-level-timestamps-from-whisper-v3s-json-is-invalid/641735
53. Qwen2 How to use fastapi to encapsulate the stream output interface · Issue #762 · QwenLM/Qwen3 - GitHub, accessed November 18, 2025, https://github.com/QwenLM/Qwen2/issues/762
54. Guide | Nginx Proxy Manager, accessed November 18, 2025, https://nginxproxymanager.com/guide/
55. Qwen/Qwen2-VL-7B-Instruct-GPTQ-Int4 - Hugging Face, accessed November

18, 2025, https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct-GPTQ-Int4