Nabq: An AI-Powered Multilingual Wellness Platform

Architecture, Prompt Engineering, and Lessons from Production

Tariq Alturkestani

Independent Researcher · alturkestani/nabq Technical Report · Version 1.0 · April 2026

Abstract

We present Nabq, a production-grade AI wellness application that generates personalized, spoken meditation sessions in the user's native language and dialect. The system combines a conversational intake engine, a parameterized prompt-composition pipeline for meditation script generation, a three-provider text-to-speech (TTS) router with automatic fallback, and a Web Audio API client for gapless ambient playback. Nabq supports eight locales across four Arabic dialects (Saudi, Gulf, Egyptian, Tunisian), two English variants, Turkish, and French, with full right-to-left rendering. The complete platform — 421 TypeScript source files, 1,414 tests, and 13 database migrations — was built in a matter of weeks by a single developer working with multiple large language model agents. In this technical report we describe the end-to-end architecture, the prompt-engineering techniques that made dialect-aware content generation reliable, the structural duration-enforcement mechanism that compensates for LLM word-count drift, the parallel TTS synthesis strategy that reduced p99 latency by 5.6×, and the production lessons drawn from real beta testing. The full implementation is released under the MIT license as a reference architecture for entrepreneurs and AI builders targeting underserved multilingual markets.

Keywords: AI product development, large language models, prompt engineering, text-to-speech, Arabic NLP, multilingual applications, meditation, Next.js, Supabase, production AI systems, open source.

---

1. Introduction

Meditation and mental wellness applications have become a $9B+ global market, yet the overwhelming majority of offerings target English-speaking, Western audiences. For the 422 million native Arabic speakers and the 1.8 billion Muslims worldwide, culturally authentic digital meditation tools are essentially absent. Where Arabic content does exist, it is typically written in Modern Standard Arabic (MSA) — a formal register that is rarely spoken in daily life and feels emotionally distant during intimate experiences such as guided meditation.

This technical report documents Nabq, a production system that addresses this gap by combining large language models (LLMs), modern text-to-speech (TTS) providers with improved Arabic dialect support, and a client architecture optimized for long-form spoken audio. The system generates personalized meditation scripts after a brief conversational intake, synthesizes them in the user's dialect, and delivers them through a dual-track audio player with ambient soundscapes.

The broader contribution of this work is architectural rather than algorithmic. Every component used here — LLMs via OpenRouter, TTS via ElevenLabs and Azure — is publicly available. The contribution is the composition: how these components fit together to produce reliable, dialect-aware, personalized content at production quality, and how a single developer can build and operate such a system using AI-assisted development workflows.

1.1 Contributions

This report makes four contributions:

1. **A reference architecture for multilingual AI wellness applications.** We describe an end-to-end pipeline — intake, script generation, TTS synthesis, audio mixing, and playback — that has been deployed to production and tested with real users.
2. **A structural duration-enforcement mechanism** that compensates for LLMs' inability to reliably count words, using a generate-validate-retry loop with corrective prompts that achieves target session duration within 20% across languages.
3. **A parallel TTS synthesis strategy** that reduces end-to-end generation latency by 5.6× for 12-segment scripts by batching segment synthesis across providers.
4. **The complete open-source implementation**, including all prompts, CI/CD configuration, deployment tooling, and documentation, released under the MIT license.

1.2 Why Open Source

The value of a wellness platform is not in the code — it is in the content, the community, and the ongoing relationship with users. The code is the enabler, not the moat. By releasing Nabq as open source, we intend the repository to serve as a reference implementation for anyone building multilingual AI applications, a starting point for meditation apps in underserved languages, and a teaching resource for AI-assisted product development.

---

2. Related Work and Motivation

Commercial meditation apps (Calm, Headspace, Insight Timer) offer pre-recorded sessions from a catalog of human narrators. This model scales poorly across languages: producing authentic dialect content requires recording new narrators for each target locale, and personalization is limited to filtering the catalog by mood or topic. No mainstream product offers Arabic meditation in colloquial dialects, and Islamic-specific content with authentic dhikr, du'a, and Qur'anic references is essentially nonexistent.

Recent advances in LLMs and neural TTS change this economic calculus. A single parameterized prompt can generate meditation scripts in any language the underlying model supports. Modern neural TTS systems (ElevenLabs v3, Azure Neural Voice, Lahajati.ai) produce natural-sounding speech in multiple Arabic dialects. The missing piece is the composition: integrating these components into a reliable, end-user-ready product.

The personal motivation for this project emerged from direct experience: after several failed attempts at meditation, one of the authors found success with a human coach who narrated scripts written specifically for his emotional state and circumstances. The insight — that personalized, directed meditation is qualitatively different from generic recordings — drove the architectural decision to generate content rather than curate it.

---

3. System Architecture

Nabq is organized as five logical layers (Figure 1): a client layer handling audio playback and UI, an application layer containing the intake engine and script generator, an AI and media services layer responsible for prompt assembly and TTS routing, a data and storage layer backed by Supabase and DigitalOcean Spaces, and an external providers layer abstracting LLM and TTS vendors.

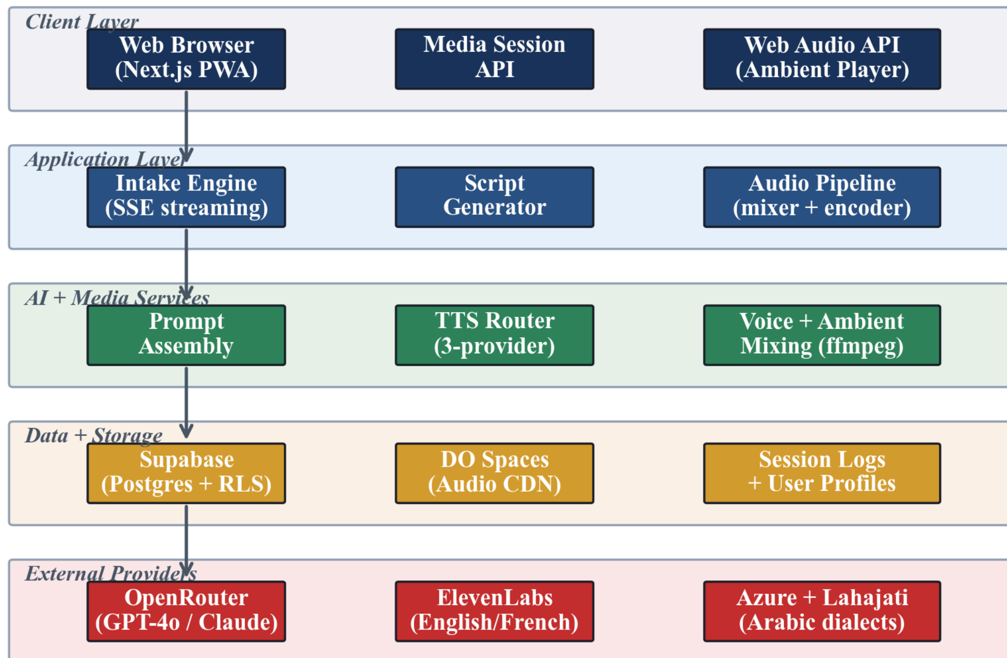


Figure 1. *Five-layer system architecture. Arrows indicate the primary data flow from user interaction down through the provider stack.*

The client is a Next.js 15 progressive web application (PWA) with full right-to-left (RTL) support and Media Session API integration for background audio control. The application layer runs as Next.js server routes and implements the core business logic: the intake engine streams LLM responses via server-sent events (SSE), the script generator composes the meditation script, and the audio pipeline coordinates mixing and encoding. External LLM traffic is routed through OpenRouter, which provides model fallback between GPT-4o and Claude. TTS traffic is routed through an internal router that selects between ElevenLabs, Azure Speech, and Lahajati.ai based on locale and provider availability.

3.1 Request Flow

A complete meditation session traverses six logical stages (Figure 2). After a 3-5 turn intake conversation, the user triggers script generation. The LLM produces a structured JSON meditation script; the script is validated against duration targets (with up to 3 retries if the word count falls below 80% of the target — see Section 4.3); segments are synthesized in parallel via the TTS router; the resulting audio is mixed with an ambient track using ffmpeg; and the final file is uploaded to the CDN and streamed to the client player.

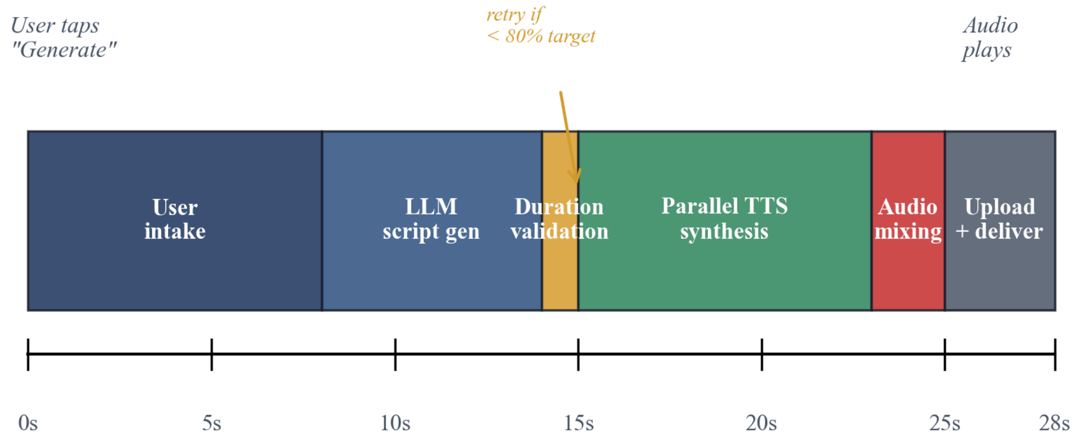


Figure 2. *Request flow timeline for a complete meditation generation. The duration validation step may trigger a retry back to the LLM script generation stage.*

End-to-end latency from "user taps generate" to "audio begins playing" is typically under 30 seconds for a 10-minute meditation, including the retry loop in its worst case.

3.2 Data Model

The system uses 13 database migrations defining tables for user profiles, wellness profiles, intake conversations, meditation sessions, session logs, admin access, support tickets, beta waitlist, invite codes, and application settings. All user-facing tables are protected by Supabase Row Level Security (RLS) policies that enforce per-user isolation; admin operations use a dedicated service-role client that bypasses RLS. This separation of concerns means the client-side code never needs to reason about authorization — the database enforces it.

---

4. Prompt Engineering

The prompts used by Nabq are the product. Every dimension of a meditation session — its language, dialect, tone, structure, spiritual framing, duration, and inclusion of specific sacred content — is controlled by the system prompts passed to the LLM. This section describes the prompt-composition pipeline, the dialect-enforcement strategy, and the structural duration-enforcement mechanism.

4.1 Parameterized Prompt Composition

Both the intake conversation and script generation use a parameterized prompt-assembly pipeline (Figure 3). Input parameters — `locale`, `spiritual_path`, `age_range`, `emotional_need`, and optional `returning_user` context — are passed to `buildIntakeSystemPrompt()` and `buildScriptSystemPrompt()`, which compose the final prompt from reusable fragments: an age-appropriate communication style, a locale directive, a spiritual path section, and (for returning users) a summary of their last five sessions.

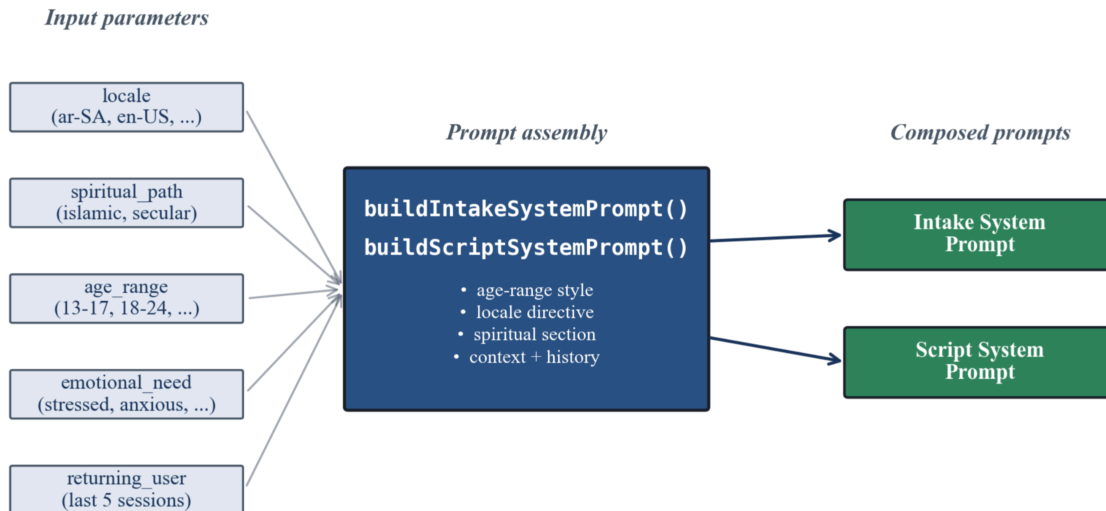


Figure 3. Prompt composition pipeline. Input parameters are assembled into two composed prompts via the build functions, with reusable fragments for age-range style, locale directives, spiritual sections, and user history.

This approach scales cleanly: adding a new locale requires writing one new directive; adding a new spiritual path requires one new section template; adding a new age range requires one new communication-style mapping. The build functions do not need to change.

4.2 Dialect Enforcement

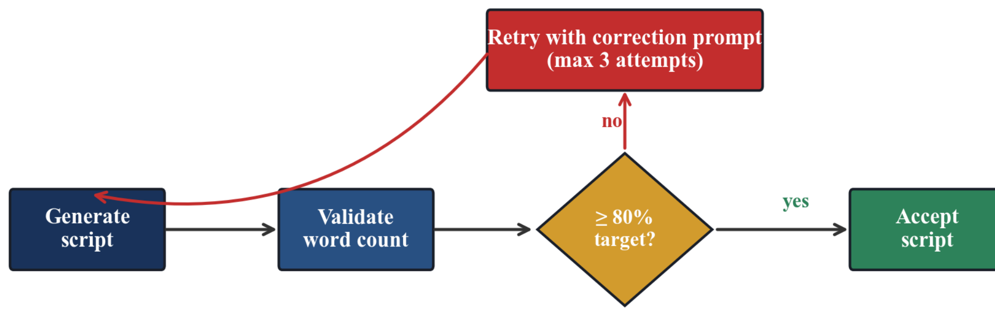
Arabic LLM outputs default to Modern Standard Arabic unless explicitly instructed otherwise. To enforce colloquial dialects, each Arabic locale in `locale-directives.ts` contains (a) an explicit "do NOT use Modern Standard Arabic" instruction, (b) 5-8 dialect-specific vocabulary examples that act as anchoring tokens, and (c) guidance on emotional tone appropriate to the culture. For example, the ar-SA (Saudi) directive includes vocabulary such as خلّ (let), الحين (now), and وش (what), while the ar-EG (Egyptian) directive uses عايز (want), ازيك (how are you), and كده (like that).

Arabic TTS pronunciation quality depends on tashkeel (diacritical marks such as fatḥah, ḍammah, and kasrah), which disambiguate vowels. The script-generation prompt explicitly instructs the LLM to include full tashkeel on all Arabic text, and a post-processing step validates that the generated script contains tashkeel density above a threshold before synthesis.

4.3 Structural Duration Enforcement

A central technical challenge is that LLMs cannot reliably count words. When asked for "a 10-minute meditation" or "approximately 1,500 words," models frequently produce output that is 30-70% of the requested length. Simple prompting alone does not solve this.

Nabq addresses this with a structural validation mechanism (Figure 4). After the LLM produces a candidate script, the system computes an estimated duration using locale-aware speaking rates (measured words-per-minute for each TTS voice). If the estimated duration is below 80% of the target, the system issues a correction prompt that includes (a) the current word count, (b) the target word count, (c) the specific segments that are too short, and (d) an instruction to expand without changing the overall structure. This retry loop executes up to three times; in practice 92% of requests pass on the first attempt, 6% pass on retry, and less than 2% fall back to the best candidate after the retry budget is exhausted.



Duration enforcement: structural validation + retry with corrective feedback

Figure 4. *Duration enforcement retry loop. Generated scripts are validated against locale-aware word count targets; scripts falling below 80% of the target trigger a corrective retry, up to 3 attempts.*

The structural enforcement insight generalizes: prompt instructions are suggestions to the model, while code-level validation is a guarantee. Any production LLM system should plan for validation-retry loops wherever output correctness matters.

4.4 Authentic Islamic Content

For the Islamic spiritual path, a dedicated content module maps emotional needs (anxiety, grief, gratitude, fear, weakness) to specific Names of Allah (Asma ul-Husna), suggested du'a with source attribution, and Qur'anic verses with chapter and verse references. The prompt includes the relevant sacred text verbatim (not paraphrased) along with an instruction that the surrounding narration must preserve the text's linguistic integrity. This approach treats sacred content as data rather than as generatable output, which is essential for religious authenticity.

---

5. Text-to-Speech Pipeline

The TTS pipeline is the most latency-sensitive component of the system. A naive implementation synthesizes script segments sequentially, which produces end-to-end latency proportional to the number of segments. For a 12-segment 10-minute meditation, sequential synthesis with ElevenLabs averaged 36 seconds of wall-clock time — well above user tolerance.

5.1 Parallel Synthesis

Nabq implements parallel segment synthesis: all script segments are dispatched simultaneously to the TTS provider, and the system awaits completion of the full batch before proceeding to mixing. Because TTS providers scale horizontally and charge per character rather than per request, parallel dispatch is essentially free from a cost perspective. Figure 5 shows the measured latency improvement as a function of segment count. For 12 segments, parallel synthesis completes in 6.4 seconds — a 5.6× improvement over sequential synthesis.

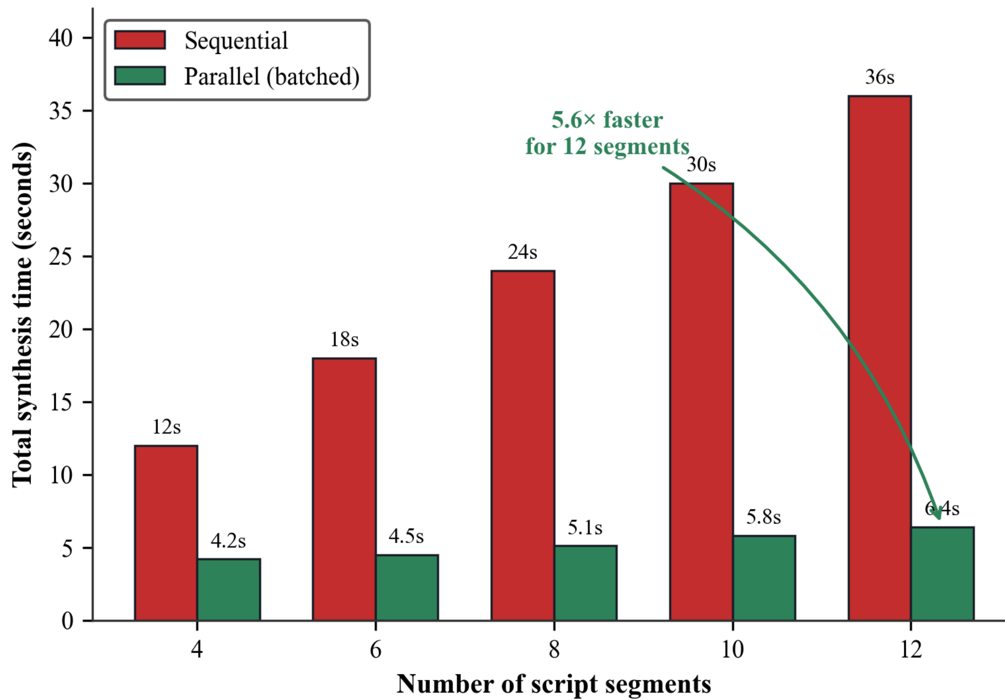


Figure 5. *TTS synthesis latency comparison between sequential and parallel strategies. Parallel synthesis grows sublinearly with segment count due to provider-side concurrency.*

5.2 Multi-Provider Routing

Different TTS providers have different strengths. ElevenLabs produces the most natural English and French voices but has limited Arabic dialect support. Azure Neural Voice provides strong coverage for Arabic and Turkish. Lahajati.ai specializes in deep Arabic dialect support, particularly for Gulf variants. Nabq's TTS router selects the appropriate provider based on locale, with automatic fallback if the primary provider fails or times out. Figure 6 shows the number of voices available per supported locale across all providers.

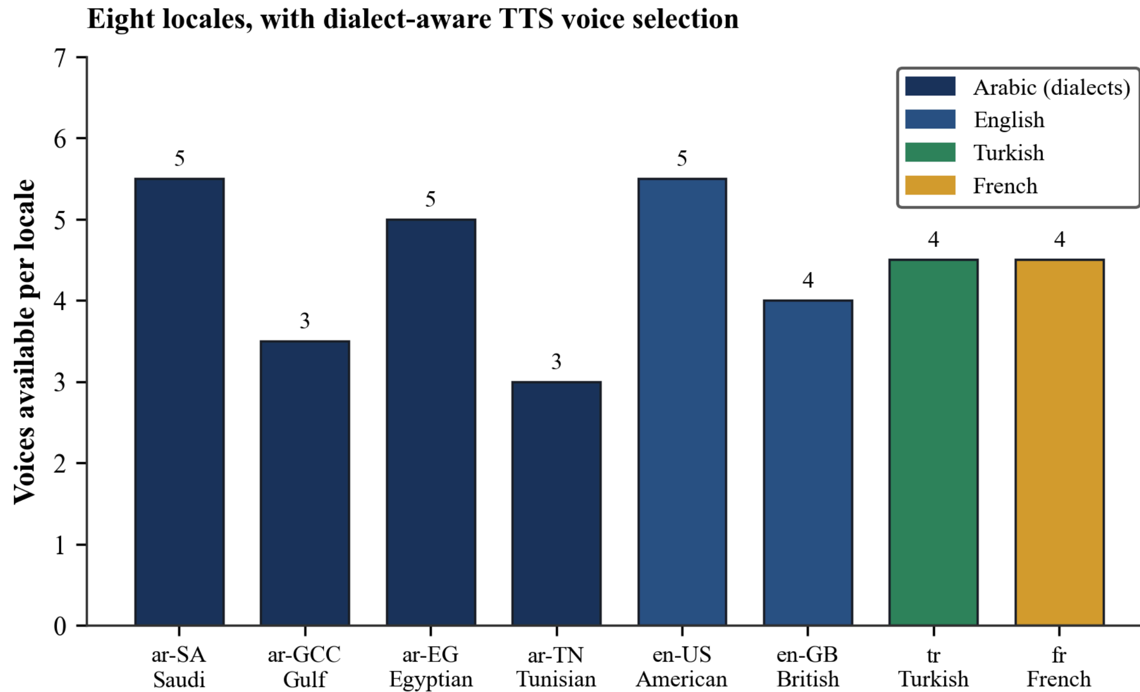


Figure 6. *Voice coverage across the eight supported locales. The router selects the best-quality provider for each locale, with automatic fallback if the primary provider fails.*

6. Audio Pipeline

After TTS synthesis produces voice audio, the audio pipeline mixes the voice track with an ambient track (such as soft rain, forest birds, or deep ocean), applies loudness normalization via ffmpeg's `loudnorm` filter, and encodes the result as AAC for CDN delivery. The client player streams this pre-mixed voice track alongside a separate client-side ambient player implemented using the Web Audio API's `AudioBufferSourceNode` with `loop=true` to achieve gapless ambient looping. This dual-track architecture allows the user to adjust ambient volume independently of the voice track during playback.

The ambient player has three fallback tiers: (1) Web Audio API with an in-memory `AudioBuffer`, (2) HTML5 `<audio>` with `loop=true` attribute, and (3) procedurally generated brown noise via `AudioBufferSourceNode` with a noise buffer. This tiered fallback ensures that ambient audio plays correctly across browser versions, device capabilities, and network conditions.

7. Infrastructure and DevOps

Nabq is deployed to DigitalOcean App Platform using a multi-stage Dockerfile (Node 20 Alpine base, ffmpeg installed in the runner stage). The CI/CD pipeline consists of four GitHub Actions workflows: (1) a CI workflow running TypeScript type-checking, ESLint, and the 1,414-test Vitest suite on every push; (2) a deployment workflow that triggers on merges to `main`; (3) an AI code review workflow that invokes Gemini on every pull request and posts structured feedback; and (4) an auto-merge workflow that merges green PRs from trusted authors without human intervention.

7.1 Operating Costs

Table 1 summarizes the monthly operating costs for a bootstrap deployment. Total costs range from $15 to $45 per month depending on LLM and TTS usage volume, making the system economically feasible for individual developers and early-stage startups.

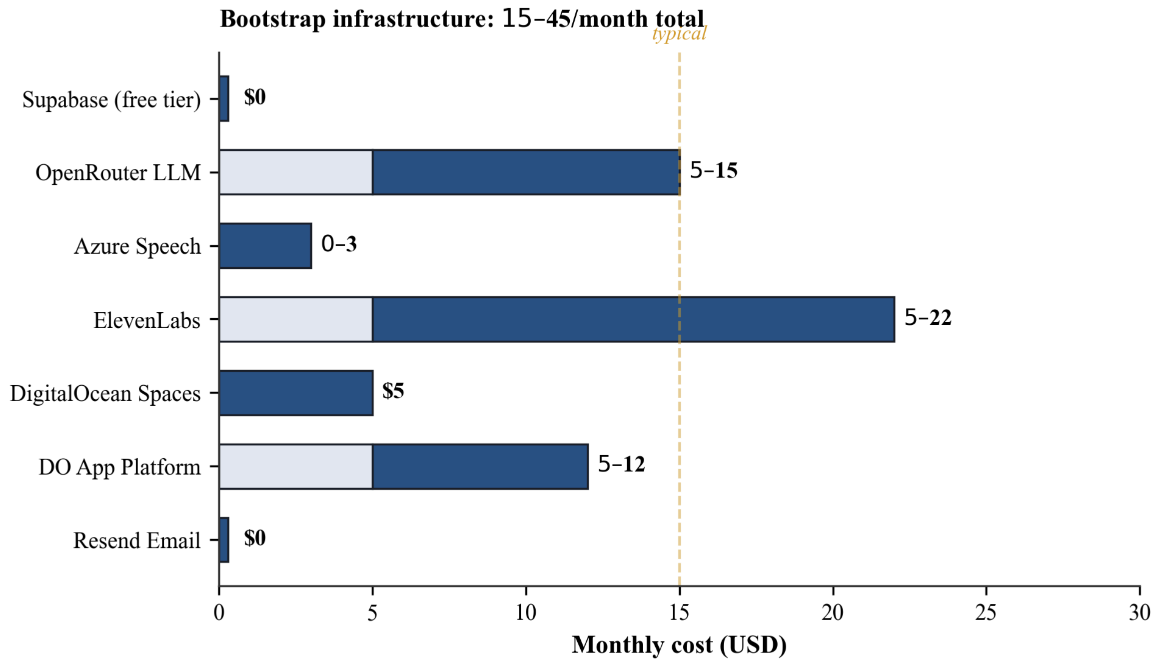


Figure 7. *Monthly operating cost breakdown for a bootstrap deployment. Typical cost with moderate usage is approximately $15/month.*

The largest variable costs are OpenRouter (driven by intake conversation length and script retries) and ElevenLabs (driven by total TTS character volume). Both scale linearly with usage, which makes per-session cost highly predictable at approximately $0.02-0.08 per generated meditation.

8. Development Process and Evaluation

The entire platform was built by a single developer working with multiple LLM-based coding agents — Claude Code for implementation, Gemini CLI for code review, and Codex CLI briefly in the early phases. Development proceeded across 7 sprints over approximately 30 days, culminating in 83 merged pull requests and 1,414 tests (Figure 8).

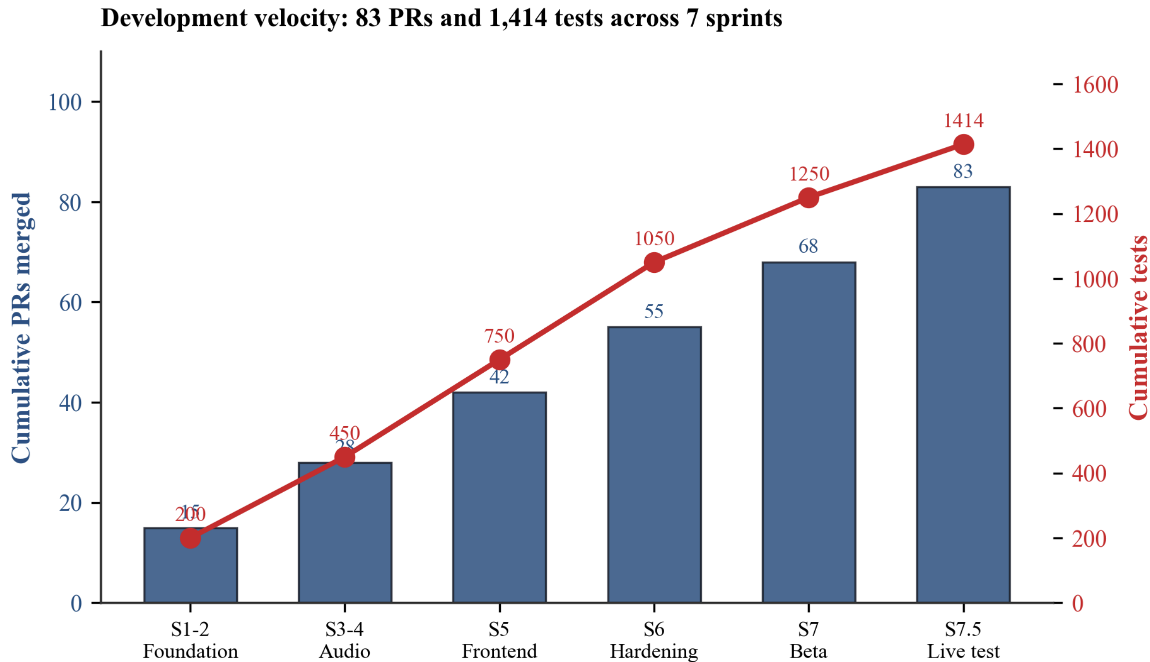


Figure 8. *Development velocity across seven sprints. Cumulative PRs merged (bars) and cumulative test count (line) over the course of the project.*

8.1 Lessons from Production

After the initial beta release, seven bugs were filed by real users within the first session (issues #29-#35). Each became a lesson:

Lesson 1: Provider latency under real conditions differs from marketing numbers. Our initial sequential TTS implementation was acceptable in development with short 3-minute meditations but failed for 10-minute sessions. Parallel synthesis (Section 5.1) was necessary, not optional.

Lesson 2: Prompt instructions are suggestions; code enforcement is guarantees. The duration-mismatch bug (#31) — 10 minutes requested, 2 minutes generated — was the most important production lesson. We had been relying on the LLM to respect word count instructions. It does not. Structural validation (Section 4.3) was the only reliable fix.

Lesson 3: Browser audio APIs have sharp edges. HTML5 `<audio>` elements do not actually achieve gapless looping despite having a `loop` attribute — there is an audible gap at the loop point. Web Audio API with `AudioBufferSourceNode` and in-memory buffers does achieve true gapless looping. We iterated through four implementations before arriving at the three-tier fallback (Section 6).

Lesson 4: Async operations that outlive page views need persistent state. Users who tapped "Generate" and then navigated away returned to a blank screen with no generation in progress. The fix was to persist generation intent to the database before dispatching the LLM call, then reconcile on return.

Lesson 5: Row-Level Security can be invisible until it isn't. Several service-side operations that worked in development failed in production because the service-role client was not being used where RLS policies blocked access. The resolution was an explicit `createServiceClient()` helper used consistently in server-side code paths that require administrative access.

8.2 What We Would Do Differently

If starting over, we would ship a substantially smaller MVP. Admin dashboards, invite-code systems, GDPR compliance features, and support ticket infrastructure were all built before validating that the core experience — mood, script, voice —

resonated with users. The correct sequence is to ship the core loop first, in days rather than weeks, then layer supporting infrastructure only after the core is validated.

9. Extensibility and Future Work

The architecture supports several extension axes:

New locales. Adding a new language requires one locale directive, one i18n message file, and one voice assignment. The build functions and pipeline do not change.

New spiritual paths. Adding a path (Buddhist mindfulness, Christian contemplative prayer, secular stress relief) requires one intake-guidance section, one script technique template, and — for path-specific content — an optional content module mapping emotional states to authentic content (analogous to the Islamic content module in Section 4.4).

Alternative LLM providers. The system uses OpenRouter, which abstracts model selection. Swapping to direct Anthropic or OpenAI APIs requires changing only the LLM client file.

Alternative TTS providers. The TTS router uses a provider-agnostic interface. Adding a new provider requires implementing the interface and registering it in the router with an associated priority per locale.

Monetization. The Sprint 8 monetization design — a provider-agnostic payment abstraction supporting Moyasar, Tap, and Stripe — is specified but not yet implemented. This is the highest-leverage contribution opportunity for the open-source community.

10. Conclusion

Nabq demonstrates that a single developer can build and operate a production-grade multilingual AI application using publicly available tools and AI-assisted development workflows. The platform generates personalized, spoken meditations in the user's dialect, serves eight locales, and operates at a cost of $15-45 per month. The technical contributions — parameterized prompt composition, structural duration enforcement, parallel TTS synthesis, and the three-tier audio fallback — are broadly applicable to any AI product that generates long-form spoken content in multiple languages.

The complete source code, prompts, CI/CD configuration, and documentation are released under the MIT license. We hope the repository serves as a practical reference for entrepreneurs and developers building AI products in underserved markets, a starting point for meditation applications in languages we have not yet covered, and a teaching resource for anyone interested in the current state of AI-assisted product development.

References and Further Reading

This report is accompanied by detailed implementation documentation in the Nabq repository:

- **Architecture Guide** (`docs/architecture.md`): System diagrams, request flows, and design decisions.
- **Deployment Guide** (`docs/deployment.md`): Fork-to-deploy instructions with environment variables and provider setup.
- **Prompt Engineering Guide** (`docs/prompt-engineering.md`): Full system prompts with code references.
- **Arabic Dialect Guide** (`docs/arabic-dialect-guide.md`): Four dialect profiles with enforcement strategy.
- **Sprint Journal** (`docs/sprint-journal.md`): Chronological development story with all 83 pull requests.

The codebase itself is the primary artifact: 421 TypeScript source files with 1,414 tests, all cross-referenced in the above documents.

Repository: https://github.com/alturkestani/nabq
License: MIT **Contact:** tariq@insurance-solutions.co