DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

Keywords: defensive-publication, prior-art, ai-agents, voice-ai, speech-to-text, text-to-speech, multi-tenant, latency-telemetry

Multi-Provider Voice Pipeline with Lazy Key Routing

A Real-Time STT → LLM → TTS Pipeline with Call-Time Per-Persona Provider Selection, Lazy Bridge-Resolved Credentials, Per-Call Stateless Lifecycle, and Three-Stage Latency Telemetry

Technical Defensive Publication — Public Prior Art

Field	Value
Title	Multi-Provider Voice Pipeline with Lazy Key Routing
Publisher / Copyright holder	Gus IT LLC (Florida, USA)
Author	Gustavo Assuncao, PhD
Publication date	2026-06-25
Version	1.0
Document type	Technical Defensive Publication (public prior art)
Classification	Public
License	AGPL-3.0-or-later (copyleft; commercial license available)
Deposit channel	To be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record.
Field	Real-time voice agents

Abstract

Production voice-agent platforms typically hardcode their speech-to-text (STT) and text-to-speech (TTS) providers, and they read provider credentials from process environment variables at start-up. As a result, switching providers (or serving two tenants with different providers from a single process) requires a code change or a process restart, and it forces every tenant's secrets to coexist in one shared environment. This disclosure describes a real-time voice pipeline that removes those constraints through a specific **combination** of mechanisms: **(1)** each of the three pipeline stages — STT, brain language-model (LLM), and TTS — is resolved **at call time** by looking up a provider in a registry keyed by a **per-persona / per-tenant configuration**; **(2)** the credential for the selected provider is fetched **lazily through a configuration bridge** at the moment of first use, never eagerly loaded into the process environment, using a memoized loader with a documented environment-variable fall-back for standalone operation; **(3)** each pipeline **instance is stateless and scoped to a single live call** — constructed when the call connects and destroyed on hang-up, so no cross-call state leaks; and **(4)** the pipeline emits **three independent latency measurements** (STT-ms, brain-LLM-ms, TTS-ms) plus a round-trip total, enabling per-stage SLA attribution across heterogeneous providers. The combination yields multi-tenant

voice in which distinct tenants run distinct provider stacks, secrets need not live in shared environment variables, and operators obtain stage-level latency visibility. This document gives the architecture, the call-time resolution and lazy-loading algorithm with pseudocode, the data model, an honest prior-art delta against Twilio Voice Intelligence, Vapi.ai, Bland.ai, Vonage, and NVIDIA Riva, and a clean-room reference implementation — all released as public prior art so the technique remains free to practice.

---

1. Executive Summary

1.1 Thesis

A voice agent does not need to bind its STT/TTS/LLM providers at build time or at process start. By treating each stage as a **registry entry selected at call time** from a **per-persona configuration**, and by resolving each provider's **secret lazily through a configuration bridge** at first use, a single voice process can serve many tenants — each with its own provider stack and credentials — while exposing **independent per-stage latency** for SLA enforcement. None of the four constituent ideas is individually novel; the **defensible contribution is their integration in a per-call stateless pipeline**, and that integration is what this document publishes as prior art.

1.2 Contributions

#	Contribution	Where
C1	Call-time, per-persona selection of STT / LLM / TTS from a provider registry	§6, §7
C2	Lazy, memoized resolution of provider credentials via a configuration bridge (with env fall-back), so secrets are never eagerly loaded into the process environment	§7.3
C3	Per-call **stateless** pipeline lifecycle: one instance per active call, destroyed on hang-up	§7.5
C4	**Three independent latency stages** (STT-ms, brain-LLM-ms, TTS-ms) + round-trip total, emitted as a telemetry event for per-stage SLA attribution	§7.6, §8
C5	A clean-room reference implementation reducing the combination to practice	§9, Appendix C
C6	An honest prior-art delta against the leading voice-agent platforms	§4, §5, Appendix A

1.3 Headline claim

> A real-time voice agent system in which audio is transformed by a speech-to-text component, the resulting text by a language-model component, and the response by a text-to-speech component, **characterized in that** each of the three components is selected **at call time** from a registry based on a **per-persona configuration resolved through a configuration bridge**, and **in that latency telemetry is emitted independently for each of the three components**.

1.4 Scope — what it is / is NOT

It IS:

- An architecture/method for *resolving* providers and credentials per call and per persona, and for *measuring* the resulting pipeline per stage.
- Provider-agnostic: STT, LLM, and TTS providers are interchangeable registry entries.

It is NOT:

- A new STT, TTS, or LLM *model* or codec — it orchestrates existing providers.
- A telephony transport — it is agnostic to the call transport (e.g. media streams or gather-style request/response); the transport is out of scope.
- A claim of *certified* compliance with any standard (see §12 for the careful language of "semantic alignment").
- A barge-in / turn-taking / VAD algorithm — those are orthogonal and out of scope.

---

2. Introduction & Motivation

2.1 The concrete problem

Consider a SaaS platform that runs AI "personas" which can answer and place phone calls. Two facts make provider binding painful:

1. **Heterogeneous quality/cost/latency across providers.** STT accuracy and TTS naturalness differ by language, accent, and domain; price and latency differ too. The "best" provider is tenant- and use-case-specific. One tenant wants Deepgram Nova for English call-center audio at low latency; another wants Whisper for multilingual fidelity; a third wants ElevenLabs voices for brand reasons and pays for it.
2. **Secrets are per-tenant.** Each provider account belongs to a tenant. If the process reads `OPENAI_API_KEY`, `DEEPGRAM_API_KEY`, `ELEVENLABS_API_KEY`, `AZURE_TTS_KEY` from its environment at boot, then **all tenants share one set of keys**, or the process must be partitioned per tenant — defeating density — and **every key sits in the process environment** whether used or not.

2.2 The "tax" of the status quo

Status-quo behavior	Cost it imposes
Provider hardcoded at build time	A code change + redeploy to switch provider, even for one tenant
Keys read from env at process start	All tenants' secrets live in one environment; rotation = restart; least-privilege impossible
One provider per process	Multi-tenant requires N processes or N deployments; poor density
Single end-to-end latency number	Cannot tell whether a slow call is STT, the LLM, or TTS; SLA disputes are unattributable
Long-lived pipeline objects	Cross-call state can leak; memory grows; per-call isolation is hard to reason about

2.3 Why existing approaches fall short

Commercial voice-agent platforms (Twilio, Vapi.ai, Bland.ai, Vonage) *do* abstract providers, and many let you pick a voice or model per assistant. But (a) the selection is generally a **configuration of a managed product**, not an in-process **registry resolved per call** that you operate; (b) **credential handling** is either the vendor's responsibility (you do not run the keys) or env-bound; the specific pattern of **lazy,**

bridge-resolved, memoized credential loading inside your own pipeline is not their concern; and (c) **per-stage latency** is exposed as analytics in some products but is not generally available as a **three-number event emitted by a per-call stateless pipeline you host**. The novelty here is in **owning and combining** these properties inside one self-hosted, multi-tenant pipeline. (See §4–§5 for the explicit delta.)

2.4 Why now

By 2026, real-time voice agents are commodity building blocks, multi-tenant SaaS is the norm, and secret-management discipline (no broad env secrets, least privilege, rotation without restart) is an audit expectation. The combination of these pressures makes the call-time + lazy-bridge + per-stage-telemetry pattern practically valuable and worth publishing before someone fences it.

---

3. Problem Statement

3.1 Formal framing

Let a voice turn be a function

```
turn : Audio_in × Persona × Config → Audio_out
```

decomposed as `tts( llm( stt( Audio_in ) ) )`. Each of `stt`, `llm`, `tts` is chosen from a registry `R_stage = { p_1, p_2, ... }` of provider implementations. A selection policy

```
select : (stage, Persona, Config) → provider
```

must run **at call time** (so persona/tenant configuration governs it) and must **not** require any provider's credential to be present at process start. A credential resolver

```
key : provider → Secret
```

must be **lazy** (invoked on first use of the provider) and **memoized**, and must fall back to environment configuration when no bridge is available. Each stage invocation must record a duration `d_stage`, and the turn must emit `(d_stt, d_llm, d_tts, d_total)` as an event. The pipeline object's lifetime must equal the call's lifetime.

3.2 Derived requirements

ID	Requirement	Satisfied in
R1	STT, LLM, and TTS providers are each selectable per call without code change	§6, §7.2
R2	Selection is driven by a **per-persona / per-tenant** configuration resolved through a bridge	§7.2, §8
R3	Provider credentials are resolved **lazily** at first use, **never** eagerly loaded into the process environment	§7.3
R4	The credential loader is **memoized** and falls back to env in standalone mode	§7.3, §9
R5	Each pipeline **instance is stateless** and scoped to exactly one active call	§7.5
R6	On call end the instance is **destroyed**, releasing any streaming connections and resetting counters	§7.5
R7	**Three independent** latency measurements (STT-ms, brain-LLM-ms, TTS-ms) are captured	§7.6
R8	A round-trip total and token usage are captured and emitted with the three stage latencies	§7.6, §8
R9	Failure of any stage is surfaced as a typed error and (where relevant) a vendor-usage event, without crashing other calls	§11
R10	The mechanism is transport-agnostic (streaming or request/response)	§6, §15
R11	No secret, hostname, or tenant identifier is required to be hardcoded	§7.3, §12
R12	Per-stage latency supports SLA attribution and alerting	§7.6, §13

4. Related Work & Prior Art

We build on, and explicitly acknowledge, a rich landscape. This invention is a *combination*, not an *ex nihilo* creation.

Source	What it provides (and we adopt the idea from)
Twilio Voice Intelligence / Media Streams	Telephony transport, streaming audio over WebSocket, provider-managed transcription. We adopt the *streaming transport* abstraction conceptually.
Vapi.ai	A managed voice-agent product letting you pick STT/LLM/TTS providers per assistant in its dashboard. Establishes that *provider choice per assistant* is desirable.
Bland.ai	Hosted voice agents with model/voice selection. Establishes user-facing provider abstraction.
Vonage Voice API / AI Studio	Provider-abstracted programmable voice.
NVIDIA Riva	A pluggable STT/TTS server with swappable models. Establishes *pluggable speech components*.
OpenAI Whisper API; Deepgram Nova-2	The concrete STT providers behind the registry entries.
Azure Neural TTS; ElevenLabs	The concrete TTS providers behind the registry entries.
12-Factor App (factor III, "Config")	The discipline of separating config from code. We *extend* it: instead of "store config in the environment," we resolve provider config/secrets through a bridge **lazily at use time**.
Service Locator / Provider Registry pattern; Dependency Injection	The structural pattern of selecting an implementation from a registry. We apply it *per call, per persona*.
Lazy initialization / memoization (classic)	The credential loader's lazy + memoized behavior.
OpenTelemetry spans / RED metrics	The discipline of per-operation latency. We specialize it to a *fixed three-stage* voice schema emitted as one event.
HashiCorp Vault / cloud secret managers	The "don't put secrets in env" principle, which the configuration bridge realizes in-process.

> **Adoption rationale.** Each of the above is a building block we deliberately reuse. The contribution is the *specific integration* (§5).

---

5. Prior-Art Delta

The table below isolates each novel feature and states, per source, what the source **has**, what it **lacks**, and what **THIS** adds.

Prior source	What it HAS	What it LACKS	What THIS adds
Vapi.ai / Bland.ai	Per-assistant provider choice in a managed dashboard	A *self-hosted, in-process* registry resolved **at call time** per persona; lazy bridge-resolved keys you own; a fixed three-stage latency event	Operator-owned call-time resolution + lazy key bridge + 3-stage telemetry in one per-call pipeline
Twilio Voice Intelligence	Streaming transport; vendor transcription/analytics	Per-persona STT+LLM+TTS *all three* swapped per call by your registry; lazy in-process key resolution	The full three-stage swap under your control with credential laziness
NVIDIA Riva	Pluggable, swappable STT/TTS models	Per-tenant config-bridge credential routing; per-call stateless lifecycle; combined 3-stage SLA event	Tenant-keyed selection + lazy keys + lifecycle + telemetry around the pluggable engines
12-Factor "Config in env"	Config/secret separation principle	**Laziness** — env is eager/global; all secrets present at boot	First-use, memoized, bridge-resolved secrets so no provider key need exist in env unless used
OpenTelemetry / RED	Generic per-operation latency	A *voice-specific fixed schema* (STT/LLM/TTS) emitted as one round-trip event tied to persona+tokens	Domain-shaped three-stage event purpose-built for voice SLA attribution
Service Locator / DI	Registry-based implementation selection	Application of the pattern *per call* keyed by *persona config from a bridge*	Per-call, per-persona registry resolution as the selection policy

Net novel combination (what the defensive publication fences): *call-time per-persona registry selection of all three stages + lazy, memoized, bridge-resolved credentials with env fall-back + per-call stateless lifecycle + three-stage latency event*, integrated in one pipeline.

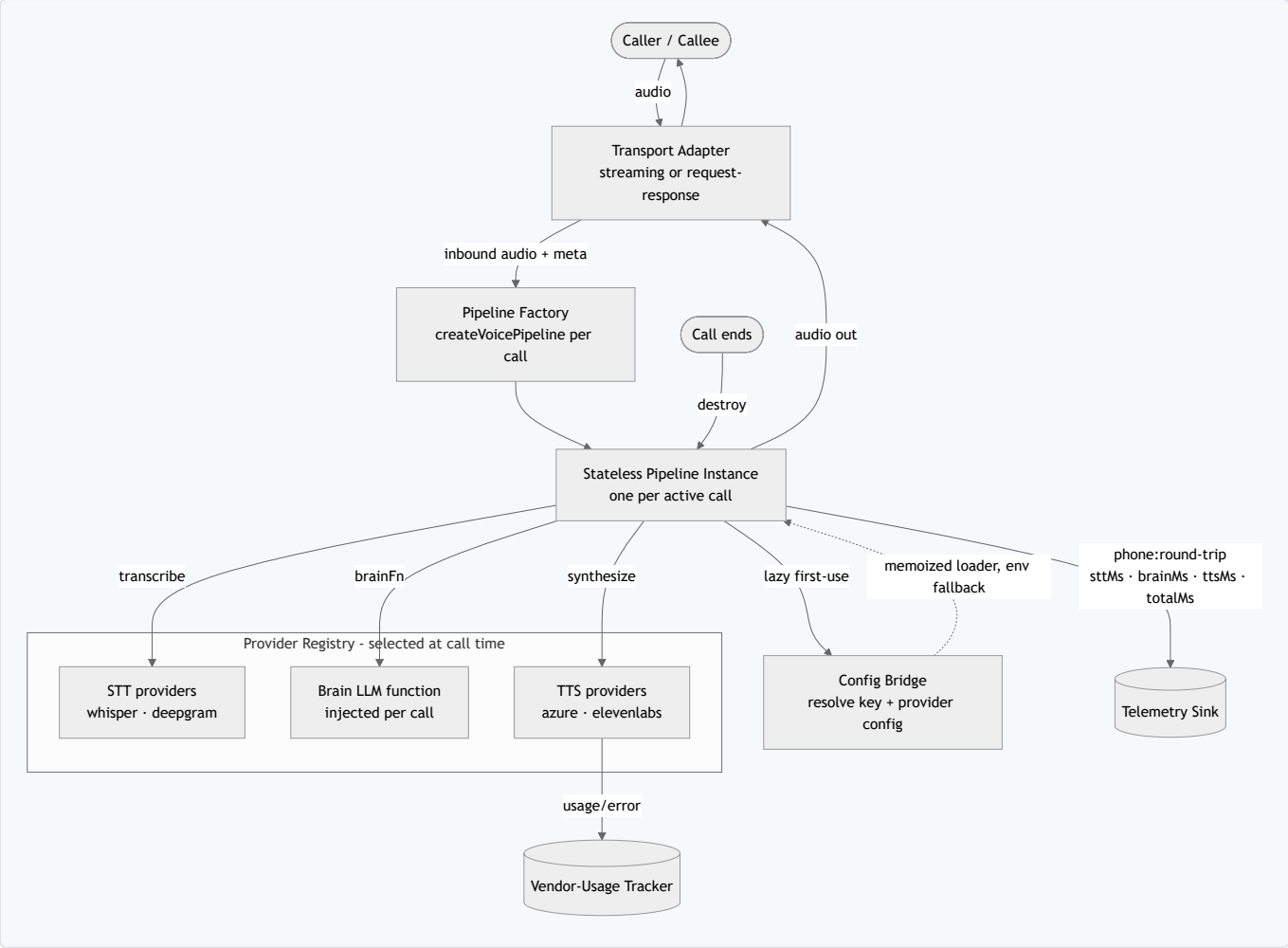
---

6. System Architecture

6.1 Components

- **Call Handler / Transport Adapter** — terminates the telephony transport (streaming media or request/response), hands audio to the pipeline and audio back. Transport-agnostic; out of scope for the claims.
- **Pipeline Factory** — `createVoicePipeline(opts)`: constructs one stateless pipeline instance per active call from `{ personaId, voiceProfile, sttProvider, ttsProvider, events, log }`.
- **Provider Registry** — maps a provider id to an implementation for each stage: `STT = { whisper, deepgram, ... }`, `TTS = { azure, elevenlabs, ... }`, plus an `LLM` (brain) function injected per call.
- **Config Bridge** — resolves effective configuration (including provider credentials) at first use; the *lazy key loader* lives here.
- **Telemetry Sink** — receives the three-stage `phone:round-trip` event.
- **Vendor-Usage Tracker** — records per-provider usage/errors (e.g. character counts) independently of the latency event.

6.2 Architecture diagram



See docs/FIGURES.md Figure 1.

6.3 Cross-cutting properties

- **Statelessness.** No cross-call state; each instance owns only its own counters, freed on `destroy()`.
- **Fail-open density.** A failure on one call's stage throws a typed error for that call only; other concurrent calls (other instances) are unaffected.
- **Config-driven.** Provider ids and credentials come from configuration / the bridge / env — never hardcoded (R11).
- **Transport independence.** The pipeline never assumes a particular transport (R10).

---

7. Detailed Mechanics

7.1 Overview of control flow

For each call: the factory builds an instance bound to a persona and provider selection; on each turn the instance runs `transcribe → brainFn → synthesize`, times each stage, and emits a single three-stage event; on hang-up the instance is destroyed.

7.2 Call-time provider selection (R1, R2)

Selection precedence (most to least specific):

1. Explicit `opts.sttProvider` / `opts.ttsProvider` passed by the call handler (which itself derives them from the persona/tenant row).
2. The persona's `voiceProfile` (per-persona JSON, e.g. which Azure voice or ElevenLabs voice id).
3. Process defaults (`DEFAULT_STT`, `DEFAULT_TTS`) sourced from configuration/env — used only when nothing more specific is provided.

The selected id indexes the registry: `whisper → transcribeWhisper`, `deepgram → transcribeDeepgram`, `azure → synthesizeAzure`, `elevenlabs → synthesizeElevenLabs`. The brain LLM is injected per call as `brainFn(personaId, userText)` so the LLM provider is equally swappable.

7.3 Lazy, memoized, bridge-resolved credentials (R3, R4, R11)

The key loader is **not** invoked at module load or at process start. It is created lazily the first time a provider that needs it is used, memoized behind a guard, and falls back to environment configuration if no bridge is reachable (standalone mode).

```
state: _getApiKey = null, _apiKeyLoaded = false

function ensureApiKeyLoader():
    if _apiKeyLoaded: return            # memoized — runs at most once
    _apiKeyLoaded = true
    try:
        bridge = import(config-bridge)  # dynamic import, lazy
        _getApiKey = async () =>
            cfg = await bridge.getEffectiveConfig()
            return cfg.providers[provider].apiKey   # bridge-resolved
    catch:
        _getApiKey = null               # standalone — fall back to env at call site

function resolveKey(provider):
    await ensureApiKeyLoader()
    if _getApiKey: return await _getApiKey()        # via bridge
    return env[KEY_FOR(provider)]                   # documented env fall-back
```

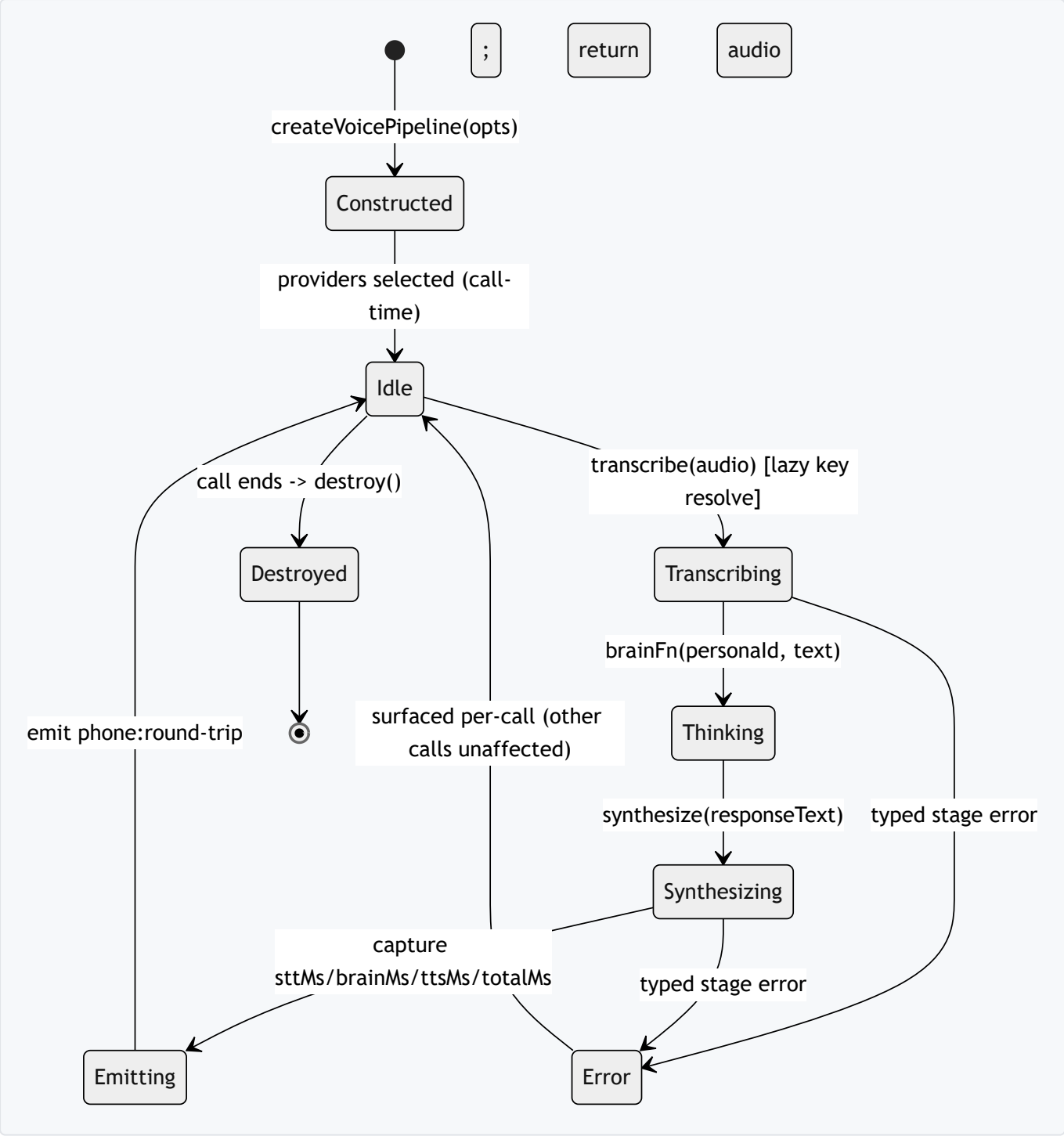
Properties:

- **Lazy:** no provider key need exist anywhere until a call actually uses that provider (R3).
- **Memoized:** the dynamic import + loader construction happen at most once per process (R4).
- **Fall-back:** standalone deployments without a bridge still work via env (R4) — but env is the *fall-back*, not the primary path (R11).
- **Per-tenant safety:** because resolution happens at call time, the bridge can return a *tenant-specific* credential for that call.

7.4 Data structures

Structure	Shape	Purpose
`opts`	`{ personaId, voiceProfile, sttProvider, ttsProvider, events, log }`	Per-call construction input
`voiceProfile`	JSON: `{ azureVoice?, elevenLabsVoiceId?, elevenLabsModel?, stability?, similarityBoost?, style? }`	Per-persona voice config
STT result	`{ text, confidence, durationMs, segments? words? }`	
TTS result	`{ audioBuffer, format, sampleRate, durationMs }`	Stage-3 output + latency
Round-trip event	`{ personaId, userText, responseText, sttMs, brainMs, ttsMs, totalMs, tokensUsed }`	The three-stage telemetry
Stats	`{ personaId, sttProvider, ttsProvider, chunksProcessed, totalTokens, totalDurationMs, avgChunkMs }`	Per-instance counters

7.5 Per-call stateless lifecycle (R5, R6)



See docs/FIGURES.md Figure 2. The instance holds only its own counters; `destroy()` releases streaming connections and zeroes counters so no state survives the call.

7.6 Three-stage latency telemetry (R7, R8, R12)

Each stage is timed independently:

```
roundTripStart = now()
stt   = transcribe(audio)              # stt.durationMs measured inside
if stt.text is empty: return early (silence)
brainStart = now()
responseText, tokensUsed = brainFn(personaId, stt.text)
tts   = synthesize(responseText)       # tts.durationMs measured inside
emit "phone:round-trip" {
    personaId, userText: stt.text, responseText,
    sttMs:   stt.durationMs,
    brainMs: now() - brainStart - tts.durationMs,   # isolate LLM time
    ttsMs:   tts.durationMs,
    totalMs: now() - roundTripStart,
    tokensUsed
}
```

Note that `brainMs` is computed by subtracting the synthesis time from the span that began at `brainStart`, isolating the language-model portion from synthesis. The three numbers (plus total) let an operator attribute a slow call to STT, the LLM, or TTS — across *different providers per tenant* — and alert per stage.

7.7 Error handling

- Each provider call checks for a resolvable credential and throws a typed, descriptive error if absent (e.g. "DEEPGRAM_API_KEY not set for Deepgram STT").
- Non-2xx provider responses throw with the provider's status and body text.
- TTS errors additionally emit a best-effort vendor-usage event (character count, voice id, model, status) so cost/usage tracking is not lost on failure.
- Errors are per-instance; because each call has its own instance, a failure cannot corrupt other concurrent calls.

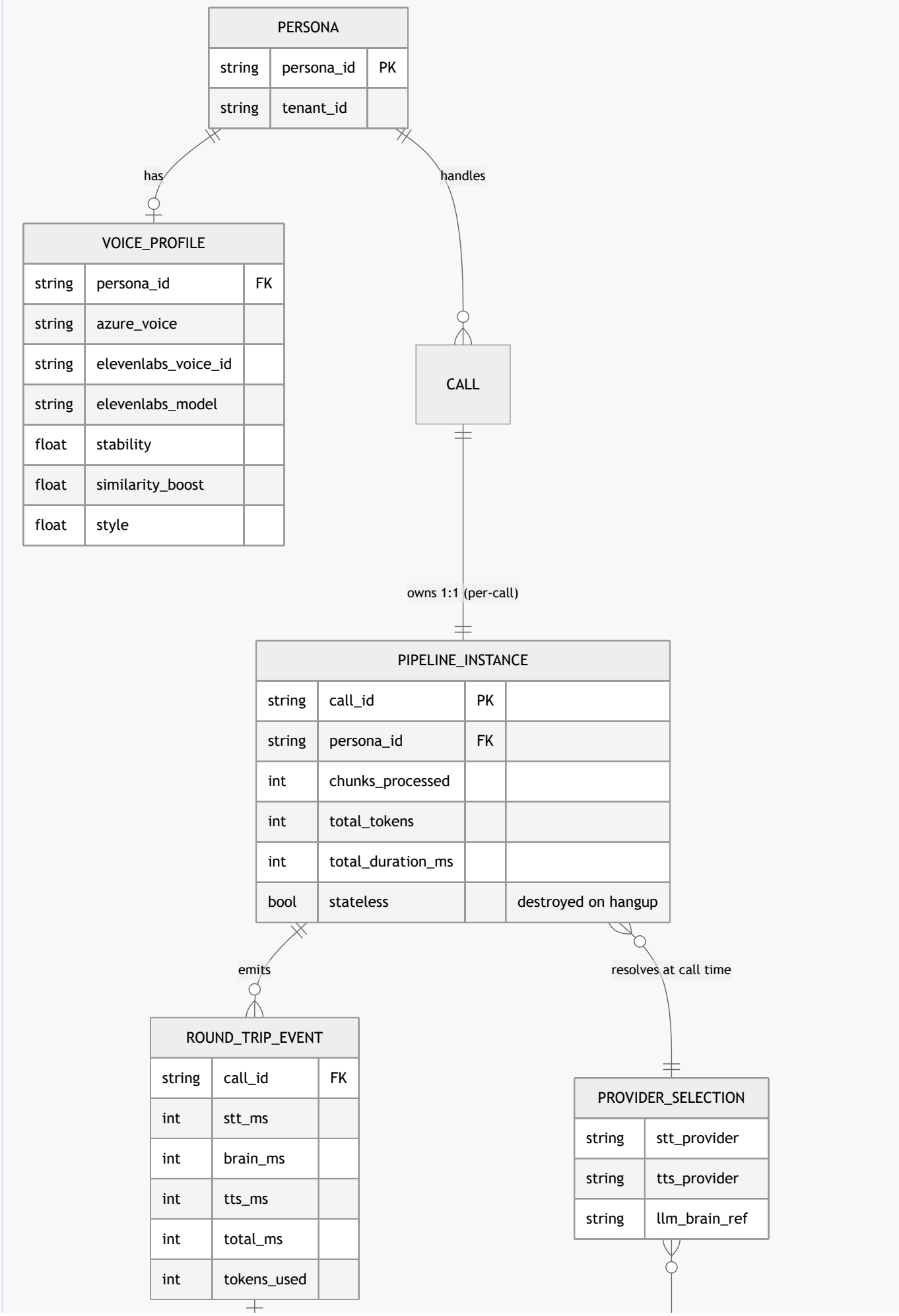
7.8 Configuration points

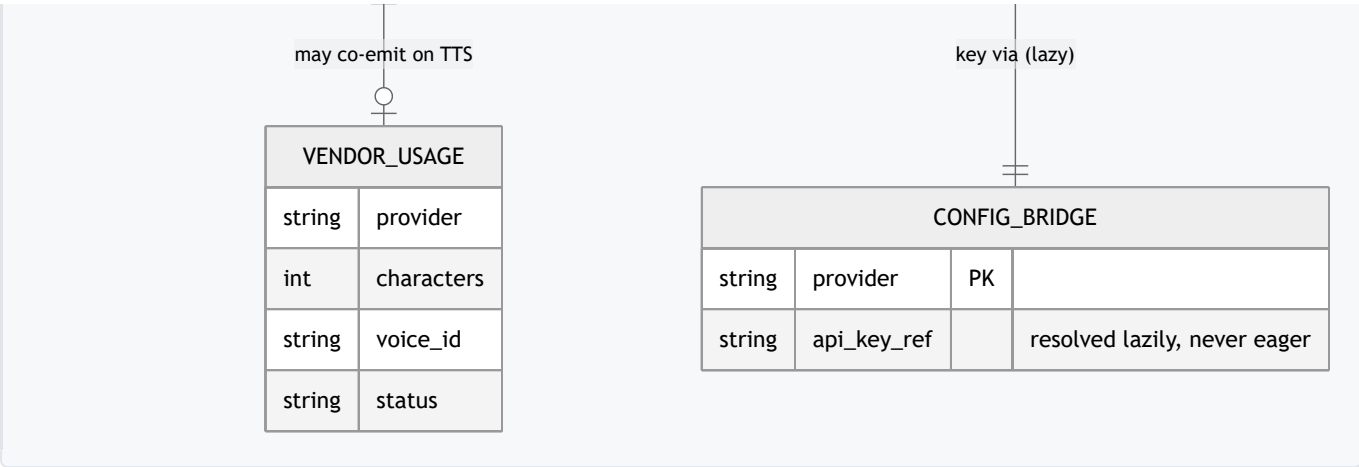
Config	Effect
`voiceProfile.*`	Per-persona voice and synthesis parameters
default STT/TTS provider	Process fall-back when no per-call selection is given
config bridge `providers.<p>.apiKey`	Primary, lazily resolved credential source
env `<PROVIDER>_API_KEY`	Documented standalone fall-back only

---

8. Data Model

The mechanism is largely in-process, but it reads a per-persona configuration and writes telemetry/usage. The conceptual entities:


PERSONA
string persona_id PK
string tenant_id
has
handles
VOICE_PROFILE
string persona_id FK
string azure_voice
string elevenlabs_voice_id
string elevenlabs_model
float stability
float similarity_boost
float style
CALL
owns 1:1 (per-call)
PIPELINE_INSTANCE
string call_id PK
string persona_id FK
int chunks_processed
int total_tokens
int total_duration_ms
bool stateless destroyed on hangup
emits
resolves at call time
ROUND_TRIP_EVENT
string call_id FK
int stt_ms
int brain_ms
int tts_ms
int total_ms
int tokens_used
PROVIDER_SELECTION
string stt_provider
string tts_provider
string llm_brain_ref



See docs/FIGURES.md Figure 3. `CONFIG_BRIDGE.api_key_ref` is deliberately a *reference resolved lazily*, never an eagerly-loaded secret. The `PIPELINE_INSTANCE` to `CALL` relationship is strictly 1:1 and ephemeral (R5/R6).

---

9. Reference Implementation & Enablement

The directory `src/` contains an **original, clean-room** Node.js implementation that reduces the combination to practice. It is illustrative — not the proprietary production code — and is dependency-light (Node standard library; providers are stubbed so it runs offline).

What `src/` demonstrates:

- **`provider-registry.js`** — STT, LLM, and TTS provider registries; lookup by id at call time (R1).
- **`config-bridge.js`** — a **lazy, memoized** credential resolver with an environment-variable fall-back (R3, R4, R11). It records whether resolution happened via the bridge or env, and proves the loader runs at most once.
- **`voice-pipeline.js`** — `createVoicePipeline(opts)` building a **stateless per-call** instance (R5, R6) that times **three stages** and emits a `phone:round-trip` event (R7, R8); typed stage errors (R9); `getStats()` / `destroy()`.
- **`demo.js`** — a runnable self-check: constructs two pipelines for two personas with *different* providers, runs a turn through each with stub providers, asserts the three latency numbers are present and the env loader ran once, and prints the events. Run with `node src/demo.js`.

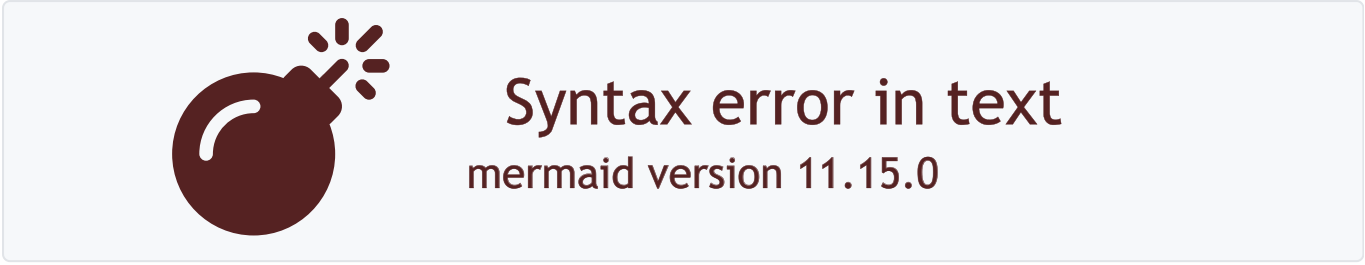
Configuration points (mirroring §7.8) are surfaced as `opts` and bridge parameters; no secret, hostname, or tenant identifier is hardcoded. See src/README.md and Appendix C.

---

10. Worked Example / Scenario

Scenario. Tenant A's persona "Reception-EN" uses Deepgram (STT) + Azure (TTS)

- an LLM brain; Tenant B's persona "Concierge-Brand" uses Whisper (STT) + ElevenLabs (TTS) + a different brain. Both are served from the same process; their keys live in the bridge, not in shared env.



See docs/FIGURES.md Figure 4. Tenant B's call runs an identical flow with Whisper + ElevenLabs and **Tenant B's** keys — concurrently, in the same process, with independent per-stage latency.

---

11. Security, Safety & Failure Modes

11.1 Posture

The credential resolution path is intentionally **fail-loud** (fail-closed) at the *stage* level: if a key cannot be resolved, the stage throws a typed error rather than silently proceeding. The *process* is **fail-open** at the *fleet* level: one call's failure does not affect other calls, because each call has its own stateless instance. The bridge resolution is lazy so an unused provider's missing key never affects an unrelated call.

11.2 Failure-mode / threat table (STRIDE-flavored)

Mode / Threat	Category	Effect	Mitigation in design
Missing provider key	Availability	Stage throws	Typed error names the provider+var; per-call isolation; bridge primary, env fall-back
Secret sprawl in env	Information disclosure	All tenants' keys in env	Lazy bridge resolution; env is fall-back only; key only fetched when used (R3)
Cross-call state leak	Tampering/Info disclosure	One call sees another's data	Per-call stateless instance; `destroy()` on hang-up (R5/R6)
Provider 5xx/timeout	Availability	One turn fails	Typed error w/ status+body; vendor-usage error event for TTS; call can retry/fallback
Mis-attributed latency	(Operational)	Unfair SLA blame	Three independent stage timers + total (R7)
Memoized loader poisoning	Tampering	Wrong key source pinned	Loader construction guarded once; bridge import isolated; env fall-back deterministic
Tenant key confusion	Authorization	Tenant A's key used for B	Resolution keyed by per-call persona/tenant config at call time (R2)
XML/SSML injection (TTS)	Tampering	Malformed/abused SSML	Escape user text before embedding in SSML (illustrated in §9 helper)
Unbounded instance growth	Availability	Memory leak	Lifecycle bound to call; `destroy()` zeroes counters/releases streams (R6)

11.3 Notes

This document withholds nothing security-critical: it deliberately recommends the bridge-resolved, lazy pattern *because* it is safer than broad env secrets. No real keys, endpoints, or tenant identifiers appear anywhere.

12. Standards & Framework Mapping

We claim **semantic alignment**, not certified compliance.

Standard / Framework	Relevant clause/idea	How this aligns (semantic)
12-Factor App	III. Config	Config/secrets separated from code; we *extend* to lazy, bridge-resolved, per-use resolution rather than eager env
OWASP ASVS	V2 (Auth), V6 (Stored Cryptography / secrets)	Secrets not broadly resident in process env; least-privilege per-tenant resolution feasible
OpenTelemetry / RED metrics	Per-operation latency, Rate/Errors/Duration	Three fixed stage durations + total emitted as one event; maps to RED Duration per stage
NIST SP 800-53 (SC-12/SC-28, AU-2)	Key management, audit events	Centralized credential resolution; per-turn telemetry event is auditable
ISO/IEC 27001 A.8 (Asset/secret handling)	Secrets management	Lazy bridge resolution narrows secret exposure surface
SOC 2 (Availability, Confidentiality)	Multi-tenant isolation, SLA evidence	Per-call isolation + per-stage latency as SLA evidence

No certification, audit, or conformance assessment is asserted. "Alignment" means the design's structure is consistent with the cited control's intent.

13. Evaluation Methodology

Numbers below are **illustrative** (they describe *how* to measure, not measured results from any deployment).

13.1 Dimensions

Dimension	Metric	Source
Latency attribution	p50/p95 of `sttMs`, `brainMs`, `ttsMs`, `totalMs`	`phone:round-trip` events
Tenancy density	Concurrent calls per process with distinct provider stacks	Instance count
Secret exposure	# provider keys resident in env at idle	Process inspection
Switch cost	Code changes needed to swap a provider for one tenant	= 0 (config only)
Isolation	Cross-call state observable after `destroy()`	Memory/state audit

13.2 Interpretation

Observation	Interpretation
`ttsMs` ≫ `sttMs`, `brainMs`	TTS provider is the bottleneck for that tenant → consider alternate TTS registry entry
Env shows 0 provider keys at idle, keys appear only on first use	Lazy resolution working (R3)
Two concurrent calls report different providers	Call-time per-persona selection working (R1/R2)
`totalMs ≈ sttMs + brainMs + ttsMs`	Telemetry self-consistent; no hidden stage

13.3 Illustrative figures (NOT measured)

Stage	Illustrative p50	Illustrative p95
STT	180 ms	420 ms
Brain LLM	600 ms	1500 ms
TTS	240 ms	700 ms
Round-trip	~1020 ms	~2600 ms

> ⚠ All numbers in §13.3 are **illustrative placeholders** to show the schema, not benchmarks from any system.

---

14. Novelty & Inventive Claims

> Drafted in patent-style prose for clarity of the prior-art record. Publishing them here is intended to **bar** later patenting of the same subject matter by others — not to assert an enforceable patent.

Claim 1 (independent). A real-time voice agent system in which an audio input is transformed by a speech-to-text component into text, the text is processed by a language-model component into a response, and the response is transformed by a text-to-speech component into output audio, *characterized in that* each of said three components is selected, **at the time a call is handled**, from a registry of provider implementations based on a **per-persona configuration resolved through a configuration bridge**, and *in that* a respective **latency measurement is emitted independently for each of said three components**.

Claim 2. The system of claim 1, wherein the credential for a selected provider is resolved **lazily upon first use** of that provider and is **not loaded into the process environment at process start**.

Claim 3. The system of claim 2, wherein the credential resolver is **memoized** such that the configuration-bridge loader is constructed at most once per process.

Claim 4. The system of claim 2, wherein the credential resolver **falls back** to an environment-variable value when the configuration bridge is unavailable, such that the system operates in a standalone mode without the bridge.

Claim 5. The system of claim 1, wherein the registry comprises at least two interchangeable speech-to-text providers and at least two interchangeable text-to-speech providers, each selectable per call without modification of program code.

Claim 6. The system of claim 1, wherein a **separate pipeline instance is constructed for each active call** and the instance is **stateless across calls**, holding only counters local to its own call.

Claim 7. The system of claim 6, wherein the pipeline instance is **destroyed upon termination of the call**, releasing any streaming connections and resetting said counters such that no state persists to a subsequent call.

Claim 8. The system of claim 1, wherein the three latency measurements comprise a speech-to-text duration, a language-model duration, and a text-to-speech duration, and the language-model duration is computed by excluding the text-to-speech duration from a measured span, thereby isolating the language-model time.

Claim 9. The system of claim 8, wherein a single telemetry event carries the three said durations together with a round-trip total and a token-usage count for the call turn.

Claim 10. The system of claim 1, wherein the per-persona configuration further specifies synthesis parameters for the selected text-to-speech provider, including at least a voice identifier and one or more of stability, similarity, and style.

Claim 11. The system of claim 1, wherein selection precedence prefers a provider explicitly specified for the call, then a provider indicated by the persona's voice profile, and then a process-default provider obtained from configuration.

Claim 12. The system of claim 1, wherein the configuration bridge returns a **tenant-specific** credential resolved at call time, such that two concurrent calls handled by the same process use credentials of two different tenants.

Claim 13. The system of claim 1, wherein a failure of any one of said three components throws a typed error scoped to the current call's instance without affecting other concurrent calls' instances.

Claim 14. The system of claim 13, wherein a text-to-speech failure additionally emits a vendor-usage event recording at least a character count, a voice identifier, a model identifier, and a status.

Claim 15. The system of claim 1, wherein the language-model component is **injected per call** as a function of a persona identifier and the transcribed text, such that the language-model provider is interchangeable on the same basis as the speech components.

Claim 16. The system of claim 1, wherein the system is **transport-agnostic**, operating identically whether audio is delivered by a streaming media transport or by a request/response gather transport.

Claim 17. A method comprising the steps performed by the system of any of claims 1–16.

---

15. Limitations & Threats to Validity

- **Narrow combination.** Each constituent (registry/DI, lazy memoization, per-call objects, per-operation latency) is individually well-known. The defensible scope is only the *integration*; an examiner could find the combination obvious (the source spec rates US ~55/100, DE/EPO ~42/100). This is precisely why we publish defensively rather than file.
- **Close prior art.** Vapi.ai/Bland.ai expose per-assistant provider choice; NVIDIA Riva is pluggable; OpenTelemetry gives per-op latency. The delta (§5) hinges on *ownership + laziness + per-call + the fixed three-stage event*.
- **Transport excluded.** We deliberately do not claim the telephony transport, barge-in, VAD, or turn-taking.
- **Illustrative numbers.** All metrics in §13 are illustrative; no benchmark of a production deployment is asserted.
- **Withheld internals.** Tenant-specific routing policies, provider-cost tables, and any production endpoints are intentionally **not** disclosed (and are not needed to enable the combination).

---

16. Future Work & Open-Source Reference App

Planned: a small open-source reference app — a multi-tenant voice gateway — that demonstrates the combination end to end with two stub tenants and pluggable real providers behind feature flags. It maps the registry, config bridge, and three-stage telemetry to a deployable service with a generic Kubernetes/AKS deployment sketch (no internal cluster identifiers). See docs/OPEN-SOURCE-APP.md. Roadmap: (1) registry + bridge + telemetry core (this repo's `src/`); (2) transport adapters; (3) dashboards for per-stage SLA; (4) provider plug-in SDK.

---

17. Conclusion

This disclosure publishes, as dated public prior art, a real-time voice pipeline that selects STT, LLM, and TTS providers **at call time** per persona, resolves their credentials **lazily through a configuration bridge** (never eagerly into env), runs as a **stateless per-call instance**, and emits **three independent latency stages**. Individually ordinary, the combination delivers multi-tenant voice with per-tenant provider stacks, reduced secret exposure, and per-stage SLA visibility. By publishing it under AGPL-3.0-or-later, Gus IT LLC keeps the technique free to practice and bars its later monopolization by others.

Appendix A — Prior-Art Landscape (well-trodden vs candidate-novel)

Well-trodden (treat as known):

- Provider abstraction / pluggable speech engines (Riva, Twilio, Vonage, Vapi, Bland).
- Service Locator / Dependency Injection / registry selection.
- Lazy initialization + memoization.
- Per-operation latency metrics (OpenTelemetry, RED).
- Secrets-out-of-code discipline (12-Factor, Vault).

Candidate-novel (the published combination):

- Call-time, **per-persona** selection of **all three** stages from an in-process registry the operator owns.
- **Lazy, memoized, bridge-resolved** credentials with documented env fall-back, so no provider key need reside in env unless used.
- **Per-call stateless** instance lifecycle (destroyed on hang-up).
- A **fixed three-stage** latency event (STT/LLM/TTS + total + tokens) for SLA attribution.

Honesty attestation. Prior-art searches behind this document are **directional, not exhaustive**. No formal freedom-to-operate opinion was obtained. Sources are named to ground the delta, not to assert completeness; the combination is published precisely so that any close prior art does not later become an *offensive* patent against practitioners.

Appendix B — Glossary

Term	Meaning
STT	Speech-to-text (transcription)
TTS	Text-to-speech (synthesis)
Brain / LLM	Language-model stage producing the response text
Persona	A configured agent identity (often tenant-scoped) driving provider/voice choice
Config bridge	A component resolving effective configuration (incl. secrets) at use time
Lazy resolution	Deferring credential/loader construction until first actual use
Memoized	Computed at most once, then cached
Per-call stateless	One instance per call, holding no cross-call state, destroyed on hang-up
Round-trip	One audio-in → audio-out turn
Three-stage telemetry	The (sttMs, brainMs, ttsMs) + total event
Registry	Map from provider id to provider implementation

Appendix C — Reference-Implementation Index

File	Purpose	Key requirements
`src/provider-registry.js`	STT/LLM/TTS registries; call-time lookup by id	R1
`src/config-bridge.js`	Lazy, memoized credential resolver with env fall-back	R3, R4, R11
`src/voice-pipeline.js`	`createVoicePipeline` — stateless per-call instance, 3-stage timing, typed errors, destroy	R5–R9
`src/demo.js`	Runnable self-check: two personas, two provider stacks, asserts telemetry + lazy loader	R1, R2, R7, R8
`src/README.md`	What it shows, how to run, clean-room disclaimer	—

Appendix D — Defensive-Publication Deposit & Timestamp

- **Publication date:** 2026-06-25.
- **Publisher:** Gus IT LLC (Florida, USA).
- **Author:** Gustavo Assuncao, PhD.
- **Version:** 1.0.
- **Deposit channel:** to be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record. No DOI is asserted at time of writing.
- **Intent:** This disclosure is **intentionally public** to establish prior art and thereby **bar later patenting of the described subject matter by others**. It is released under AGPL-3.0-or-later with its express patent grant.
