Concurrency Benchmarking for voice AI auto-AGENTS™: Azure Based Architecture and Operational Readiness

Commerce.AI

Feb 15, 2026

1 Executive Summary

This document defines a repeatable benchmarking framework and reference architecture for evaluating Commerce.AI voice AI workloads on Microsoft Azure. The intended use is technical due diligence, procurement review, security assessment, and operational planning rather than marketing collateral.

The focus is high-concurrency voice agent operation, including deployment patterns intended to support 10,000 concurrent calls at cluster level, subject to measured infrastructure limits, third-party service quotas, telephony integration characteristics, and workload design.

The benchmark framework described here is designed to answer four practical questions:

- Can the end-to-end system sustain the required number of concurrent live calls without materially degrading latency, transcription quality, or containment performance?
- Which subsystems become limiting factors first, including telephony ingress, speech services, orchestration services, large language model (LLM) endpoints, downstream APIs, logging pipelines, or escalation paths?
- Are scaling and failure behaviors acceptable for enterprise production use, including healthcare and regulated workloads?
- Is the benchmark repeatable enough to support architecture review boards, RFP responses, and operational readiness sign-off?

No benchmark results are fabricated in this paper. All result fields are intentionally represented as sample fields intended to be replaced with measured values from formal test execution.

As an illustrative sizing check, the rough capacity math in this paper supports a 10,000-call planning assumption at cluster level when the orchestration tier is deployed as a horizontally scaled thin control plane. The same math does not support making that claim for a single Standard_E4bs_v5 orchestration node. For richer real-time workflows, per-node counts fall toward the low hundreds; materially higher counts depend on keeping the node focused on control-plane duties while scaling horizontally across the cluster.

2 Scope and Assumptions

This whitepaper covers benchmark planning and execution for AI-driven voice interactions operating on Azure-based infrastructure. It assumes a production-style architecture in which telephony or contact center traffic is routed into a voice AI application stack composed of speech services, conversation orchestration, LLM inference, enterprise API integrations, observability tooling, and optional transfer to live agents.

The framework assumes the following:

- Benchmarking is performed in a controlled pre-production or isolated production-like environment.
- Azure services, model endpoints, and network paths used in the test are representative of the intended production deployment.
- The 10,000-concurrent-call position in this paper is a cluster-level claim that must be tied to the specific workload profile, language mix, prompt design, API dependencies, escalation rates, and measured results under test.
- External system limits, including CCaaS platform quotas, speech service quotas, LLM throughput quotas, and downstream API rate limits, are explicitly documented as part of the benchmark record.
- Codec, language, prompt version, model deployment, token budget, and turn-density assumptions are documented before test execution.
- Security, privacy, business associate agreement (BAA), and retention settings used in the test match the intended production control posture when regulated data or healthcare workflows are in scope.

Out of scope: commercial pricing, comparative vendor rankings, and unsupported generalizations across different model providers or telephony environments.

3 Why Concurrency Benchmarking Matters for AI Voice Agents

Concurrency testing for voice AI is materially different from testing chat or web transactions. Each active call creates a continuous stream of stateful interactions involving audio ingress, speech recognition, conversational logic, model inference, optional retrieval or API calls, text-to-speech output, and real-time interruption handling. Degradation in any layer can become audible to the caller within seconds.

For enterprise buyers, concurrency benchmarks help determine whether the platform can:

- maintain acceptable turn latency during sustained peak periods;
- absorb burst events without uncontrolled queue growth;

- protect service quality when dependencies slow down or partially fail;
- preserve compliance controls such as redaction and audit logging under load; and
- transfer gracefully to human-assisted workflows when automation thresholds are exceeded.

In healthcare and other regulated environments, concurrency benchmarking is also a governance exercise. It validates that privacy controls, logging policies, and escalation procedures behave predictably at the same operating levels expected in production.

4 Reference Azure Architecture

The reference design below is intentionally technology-pattern oriented. Specific Azure services may vary based on Commerce.AI implementation choices, customer network requirements, and approved model hosting patterns.

Figure 1 illustrates one reference deployment pattern suitable for concurrency testing on Azure.

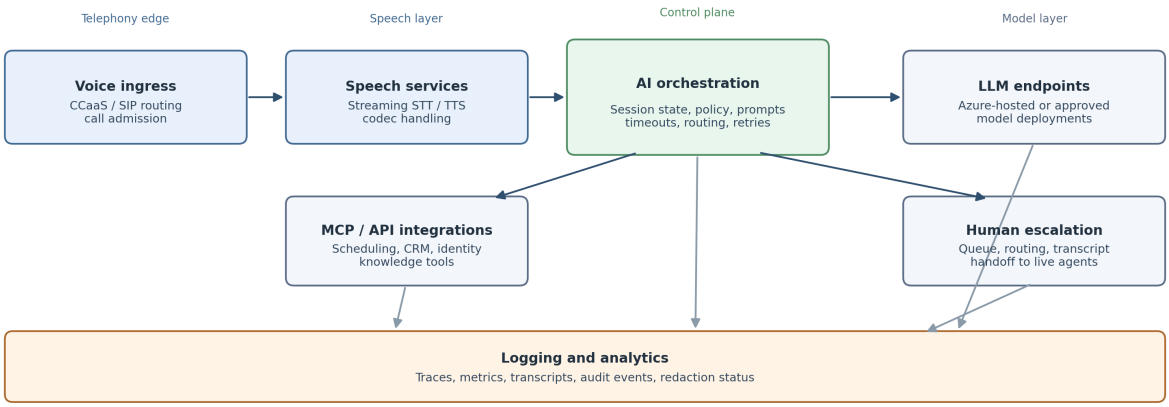

Figure 1: Reference Azure voice AI architecture for concurrency benchmarking.

Table 1 summarizes selected public Azure service characteristics relevant to concurrency planning. These are not deployment commitments; they are planning inputs that should be validated against the specific subscription, region, SKU, and quota approvals used for testing. Azure guidance emphasizes balancing reliability, security, operational excellence, and performance efficiency in workload design.[1]

[1] Microsoft Azure Well-Architected Framework overview

Table 1: Selected public Azure service characteristics relevant to voice AI benchmarking

Layer	Publicly Documented Characteristic	Benchmark Implication	Source
Speech-to-text	Azure Speech S0 documents a default limit of 100 concurrent real-time speech-to-text requests per resource, adjustable for Standard resources.	A 10,000-call target requires quota planning across multiple speech resources and, in many cases, multiple regions before load execution.	Azure Speech quotas
Text-to-speech	Azure Speech S0 documents 200 transactions per second by default for real-time text-to-speech, adjustable up to 1,000 TPS for Standard resources.	TTS rate ceilings can become limiting during dense turn-taking or simultaneous prompt playback.	Azure Speech quotas
LLM throughput	Azure OpenAI quota is scoped per subscription, region, and model or deployment type rather than tenant-wide.	Capacity planning should account for per-region quota pools, model mix, and deployment distribution.	Azure OpenAI quotas
Autoscale	Azure Monitor autoscale supports metric-based, schedule-based, and predictive scaling patterns for supported resources such as VM Scale Sets.	Load tests should include both gradual ramps and predictable peak windows to verify scale timing.	Azure Monitor / VMSS
Monitoring	Azure Monitor Logs can be zone-redundant in supported regions, and workspace-based Application Insights is the model aligned with those capabilities.	Telemetry durability and query continuity should be evaluated together with application resilience.	Azure Monitor Logs
Zone design	Azure Well-Architected guidance recommends deciding explicitly between availability-zone and multi-region strategies based on business requirements and tradeoffs.	Benchmark evidence should state whether results represent single-zone, multi-zone, or multi-region conditions.	Well-Architected reliability

Public sources for Table 1 include Azure Speech quotas and limits, Azure OpenAI quota guidance, Azure Monitor availability-zone documentation, and Azure Well-Architected reliability guidance.[2][3][4][5]

[2] Azure Speech quotas and limits
[3] Azure OpenAI quota guidance
[4] Azure Monitor availability zones
[5] Azure regions and availability zones guidance

Cloud Portability Note

This paper uses Azure service names, quotas, and VM SKUs because the reference architecture is Azure-based. The sizing method itself is portable: the same CPU, memory, network, dependency-quota, and token-throughput calculations can be mapped to equivalent 4 vCPU / 32 GiB and 8 vCPU / 64 GiB instance shapes on other cloud platforms. Any such translation should revalidate managed STT/TTS limits, LLM quota allocation, observability behavior, private-network latency, and regional availability rather than assuming Azure-specific limits carry over unchanged.

4.1 Voice Ingress / CCaaS Integration

Inbound or outbound call traffic enters through a telephony or contact center platform integrated with the Commerce.AI application layer. Typical patterns include SIP trunking, programmable voice services, or CCaaS connectors. The ingress layer should support session admission control, request correlation IDs, regional routing, and fallback handling when downstream AI components are degraded.

The benchmark record should disclose the audio path and codec used for each scenario. For example, a G.711 μ-law telephony path and an Opus wideband partner integration can produce different speech-recognition behavior, packet timing, and perceived latency even when the application logic is unchanged.

Key design considerations:

- deterministic call routing into the correct tenant, workflow, or campaign;
- call-level metadata capture for audit and troubleshooting;
- bounded retries to avoid duplicate session creation; and
- controlled failover to recorded messages, queueing, or live-agent transfer.

4.2 Speech Processing

Speech processing includes streaming speech-to-text, optional diarization or barge-in support, language handling, and text-to-speech generation. The benchmark should reflect real codec choices, packet timing, and speech segment sizes used in production because these directly affect latency and throughput.

The speech layer is often the first place where high-concurrency assumptions break if quotas, stream counts, or regional capacity are not validated in advance.

Formal reports should state how many STT and TTS resources, regions, and approved quota increases were used. Without that disclosure, latency and concurrency results are difficult to compare across benchmark cycles.

4.3 AI Orchestration Layer

The orchestration layer manages session state, tool selection, prompt construction, turn control, policy enforcement, retry logic, and response assembly. This is commonly implemented using stateless compute nodes backed by low-latency state stores, queues, and configuration services.

For concurrency testing, this layer should expose metrics for active sessions, requests per second, queue depth, per-turn processing time, and failure reason codes. It should also enforce timeouts so slow dependencies do not indefinitely block live calls.

4.4 LLM Layer

The LLM layer may use Azure-hosted models, provisioned throughput, or approved model-serving endpoints. Benchmarking should isolate model inference latency from total turn latency and distinguish between prompt-processing delay, model-generation delay, and network overhead.

Relevant factors include prompt size, tool-calling frequency, token budgets, concurrency limits per deployment, and the degree to which inference workloads are parallelized or serialized.

From a planning perspective, LLM load scales with total token throughput rather than request count alone. A useful first-order estimate is

$$TPM_{\text{llm}} = C_{\text{calls}} \times R_{\text{bot/min}} \times T_{\text{effective/response}},$$

where $T_{\text{effective/response}}$ should include the full input and output footprint of each bot response: system instructions, retained conversation context, retrieved documents, tool-call scaffolding, safety wrappers, and generated tokens. In voice applications, confirmations, interruptions, reprompts, and fallback turns often increase token demand even when the audible reply sounds short.

For that reason, benchmark planning should track not only model latency, but also median and p95 input tokens, output tokens, total tokens per turn, and context-window growth over the life of a call. Small prompt changes that improve accuracy can double or triple total TPM at the same call concurrency if turn density remains constant.

4.5 API Integration Layer

Voice agents frequently depend on downstream systems such as patient scheduling, CRM, eligibility verification, order status, identity services, or knowledge APIs. The integration layer must therefore be benchmarked as part of the real system rather than treated as an external abstraction.

This layer should implement rate limiting, circuit breaking, caching where appropriate, structured error translation, and graceful fallback responses when source systems are unavailable or slow.

4.6 Logging and Analytics

Operational telemetry should include call lifecycle events, turn-level latency, dependency timing, escalation events, redaction status, and error categorization. Logging must be sufficient for root-cause analysis while minimizing exposure of sensitive content.

The benchmark must account for the performance cost of observability itself, particularly when synchronous logging, transcript persistence, or analytics enrichment pipelines are enabled.

4.7 Escalation to Human Agents

The architecture should support controlled transfer from automation to human agents when confidence thresholds, policy rules, customer preference, or dependency failures require it. Benchmarking should include escalation under load because transfer workflows often depend on additional queues, desktop integrations, or transcript handoff operations that are not stressed during isolated AI-only testing.

5 Test Dimensions

The benchmark should vary one dimension at a time where possible, followed by combined worst-case scenarios.

Concurrent Calls

Measure low, medium, high, target, and stress conditions, for example 250, 500, 1,000, 2,000, 5,000, 7,500, and 10,000 concurrent calls. Include both sustained concurrency and short-duration burst loads.

Call Duration

Test representative mixes of short informational calls, medium transactional calls, and longer exception-handling calls. Average call duration materially affects active session counts and memory pressure.

Turn-Taking Frequency

Different workloads produce different turn densities. Appointment confirmation and FAQ flows may be sparse; troubleshooting or intake workflows may generate more frequent turns and more API calls per minute.

Codec, Language, and Prompt Profile

Benchmark scenarios should lock the codec, language set, prompt version, retrieval policy, model deployment, and context-window strategy before execution. These variables can materially change both latency and token consumption, so they should not drift between concurrency tiers.

LLM Response Time

Model latency should be measured independently and in end-to-end context. Benchmarks should record average, p95, and worst-observed response times under each concurrency tier.

API Latency

Downstream APIs should be tested at baseline, degraded, and failure-state latency bands. This is especially important when business workflows require synchronous record lookup or transaction commits.

Redaction Overhead

If PHI or other sensitive data is processed, transcript redaction and log scrubbing should be enabled during testing to measure computational and latency overhead in realistic conditions.

Logging Overhead

Measure the difference between minimal telemetry and full operational logging. Logging overhead can be nontrivial at high call counts, particularly if event enrichment or indexing occurs inline.

Escalation Behavior

Include scenarios with a 10% to 20% escalation rate to verify queue behavior, transcript handoff, routing metadata transfer, and abandonment risk during handoff.

6 Benchmark Methodology

Figure 2 shows an illustrative benchmark sequence with warm-up, steady-state, burst, and degraded-dependency phases. The exact numbers should be customized to the approved test plan.

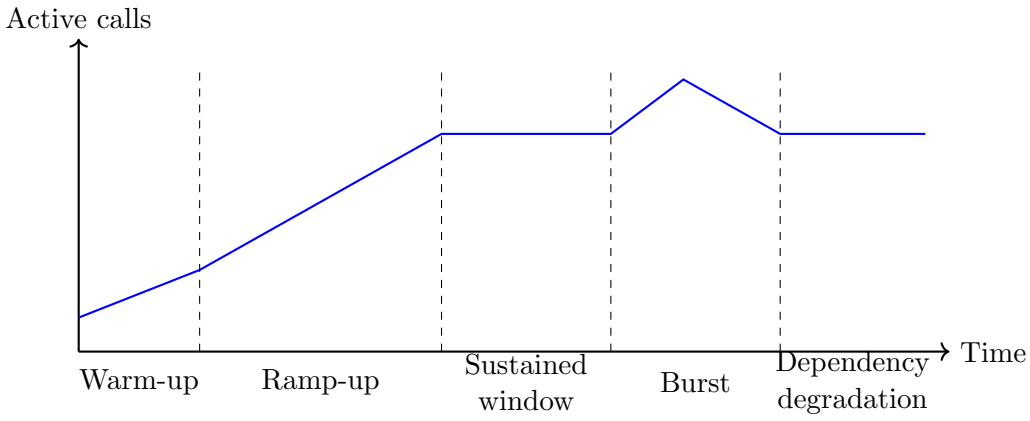

Figure 2: Illustrative concurrency benchmark sequence.

6.1 Load Generation

Load generation should emulate realistic audio and turn patterns rather than synthetic request-only traffic. Test harnesses should simulate concurrent calls with representative speech cadence, caller silence, interruption, DTMF where applicable, and API invocation patterns. If prerecorded audio is used, the corpus should include realistic variation in speaker pace, phrasing, background conditions, and language mix.

Each run should preserve a compact scenario manifest that records codec, language, model deployment, prompt version, retrieval settings, token budget, API mix, escalation target, region, and quota approvals. The manifest is what makes later benchmark comparisons auditable rather than anecdotal.

6.2 Ramp-Up Pattern

The ramp profile should be documented explicitly, for example:

- start at 100 calls;
- increase by 100 calls every 5 minutes; and
- continue until the target concurrency tier or defined stress ceiling is reached.

Ramp-up should be gradual enough to expose autoscaling behavior and dependency saturation points, not merely crash the system with instantaneous demand.

6.3 Sustained Concurrency Window

Each primary concurrency tier should be held for a sustained observation period, such as 45 to 60 minutes, so that delayed effects become visible. These include memory growth, connection pool exhaustion, logging backpressure, and autoscaling stabilization delays.

6.4 Peak Burst Testing

Separate burst tests should inject demand spikes above the steady-state level to evaluate admission control, queuing behavior, and recovery time. Burst scenarios should be short, measurable, and repeatable. Unless the approved test plan specifies otherwise, a useful default is a 1.75× steady-state demand spike held for 90 seconds, followed by observation until latency, error rate, and queue depth return to the accepted steady-state bands.

6.5 Failure-Mode Testing

Benchmark plans should include controlled failures such as instance termination, unavailable model endpoints, queue delay, storage throttling, or zone impairment. The goal is not only to measure raw throughput, but also to verify graceful degradation.

At minimum, the evidence package should include one instance-loss test, one zone or regional impairment simulation where feasible, one LLM endpoint degradation scenario, and one downstream API timeout scenario. For each case, record admission-control behavior, fallback messaging, transfer success, recovery time, and whether audit and redaction controls remained enabled.

6.6 API Degradation Simulation

Downstream APIs should be artificially slowed or error-injected to test timeout handling, fallback responses, customer messaging, and escalation triggers. Typical simulation bands include 1.5-second, 3-second, and 5-second response-time bands and 2%, 5%, and 10% error rates.

7 Metrics to Report

At minimum, formal benchmark reports should capture the following for each scenario:

- **Active calls:** observed active sessions over time.
- **Peak concurrent calls:** highest stable simultaneous call count observed.
- **Average and p95 latency:** end-to-end turn latency, speech latency, orchestration latency, and LLM latency.
- **API response time:** average and p95 latency for each critical dependency.
- **Error rate:** failed turns, failed calls, dependency errors, and internal exceptions.
- **Abandonment / fallback rate:** callers dropped, fallback messaging invoked, or self-service unable to proceed.
- **Escalation rate:** transfers to human agents, segmented by cause if possible.

- **Autoscaling events:** scaling triggers, scale-out counts, stabilization times, and any scale failures.
- **Scenario manifest:** codec, language set, model version, prompt version, token budget, region, quota approvals, and private-network posture.

Additional useful metrics include queue depth, token consumption, compute saturation, memory utilization, connection pool utilization, transcript persistence lag, and log pipeline throughput.

8 Recommended Acceptance Thresholds

Acceptance thresholds should be tuned to the business workflow, but a defensible enterprise framework typically defines thresholds in the following categories:

Category	Example Threshold	Notes
End-to-end turn latency	under 1.5 s average / under 3.0 s p95	Define separately for routine and complex turns.
Speech recognition latency	under 500 ms average / under 1.2 s p95	Measure on production codecs and language set.
LLM inference latency	under 1.2 s average / under 2.5 s p95	Separate model time from orchestration overhead.
Critical API latency	under 800 ms average / under 2.0 s p95	Include timeout and retry budgets.
Call failure rate	less than 1.0%	Define what counts as failure versus controlled fallback.
Fallback / abandonment rate	less than 3.0%	Track by scenario and cause.
Escalation success	at least 98%	Measure completed transfer, not just attempted transfer.
Recovery from burst or fault	within 10 minutes	Time to return to normal latency/error bands.

Thresholds should be approved before test execution so benchmark outcomes are interpreted against predefined criteria rather than post hoc expectations.

9 Benchmark Evidence Requirements and Caveats

The tables and figures in this paper define a benchmark framework and planning model. They should not be read as production measurements unless accompanied by a benchmark evidence package that includes raw time-series data, test-harness configuration, scenario manifests, quota approvals, dependency logs, and incident notes from failure-mode execution.

Latency, error rate, and capacity outcomes are sensitive to variables that often change between customers and benchmark cycles: codec, language mix, prompt length, model deployment, retrieval policy, API latency, escalation rate, private-network configuration, and observability settings. A

valid benchmark report should state which variables were held constant and which were intentionally varied.

For public sharing, the safest interpretation is that this paper documents the method and acceptance criteria for high-concurrency voice AI readiness. Current measured latency and throughput should be reported from the latest controlled benchmark cycle rather than inferred from the illustrative planning math in this document.

10 Illustrative Capacity Math for a 10,000-Call Cluster Target

This section provides a planning estimate, not measured benchmark evidence. Its purpose is to show how a 10,000-active-call cluster target can be supported under the stated architecture assumptions before formal load testing begins.

Microsoft Learn lists Standard_E4bs_v5 at 4 vCPU and 32 GiB RAM. If the intended target is 64 GiB per orchestration node, the closer Ebsv5 size is Standard_E8bs_v5 at 8 vCPU and 64 GiB RAM. The rough math below therefore models both the named E4bs_v5 size and the likely 64 GiB intent represented by E8bs_v5. Microsoft Learn also lists a maximum network bandwidth of 12,500 Mbps for both sizes.[6]

The model assumes that Azure Speech and Azure OpenAI remain external managed services and that orchestration nodes primarily handle session orchestration, prompt assembly, policy checks, API fan-out, telemetry buffering, and transfer control. It also assumes no media proxying on the orchestration nodes, externalized session state, and asynchronous transcript and log persistence. Under that thin-control-plane assumption set, the claim is not that one small VM must carry 10,000 calls, but that the cluster can carry 10,000 calls with horizontal scaling.

Planning Equations

The rough capacity model uses the following formulas:

$$
\begin{aligned}
C_{\text{cpu}} &= \frac{vCPU \times 1000 \times U_{\text{cpu}}}{m_{\text{cpu/call}}}, \\
C_{\text{mem}} &= \frac{RAM_{\text{GiB}} \times 1024 \times U_{\text{mem}}}{m_{\text{MiB/call}}}, \\
C_{\text{net}} &= \frac{BW_{\text{Mbps}} \times U_{\text{net}}}{b_{\text{Mbps/call}}}, \\
C_{\text{effective}} &= \min\left(C_{\text{cpu}}, C_{\text{mem}}, C_{\text{net}}\right).
\end{aligned}
$$

In this planning model, sustained CPU is capped at 65%, application memory at 70%, and control-plane network bandwidth at 50% to preserve operational headroom for bursts, failover, and noisy-neighbor effects.

[6] Microsoft Learn: Ebsv5-series Azure VMs

Thin-node Assumption	Value	Interpretation
Bot responses per minute	6	Balanced conversational cadence rather than dense interrupt-driven speech.
User response turns per minute	6	Roughly one user response per bot response.
Control events processed on node	12 per minute	Session updates, policy checks, routing logic, and prompt assembly events.
CPU cost per control event	2.5 ms	Event-driven handler with external speech, LLM, and state services.
Steady CPU per active call	30 ms/min = 0.5 ms/s	Basis for the platform-target thin-node profile used below.

Table 3: Illustrative decomposition behind the platform-target thin-node assumption

Parameter	Standard_E4bs_v5	Standard_E8bs_v5	Planning Note
vCPU	4	8	Used as the basis for sustained CPU planning budget.
RAM	32 GiB	64 GiB	E4bs_v5 is the named SKU; E8bs_v5 is the closer 64 GiB analog.
Published max network bandwidth	12,500 Mbps	12,500 Mbps	The rough math reserves only 50% of this for steady-state planning.
Planning CPU budget	2,600 ms/s	5,200 ms/s	Equals vCPU $\times$ 1000 $\times$ 65%.
Planning application memory budget	22.4 GiB	44.8 GiB	Equals published RAM $\times$ 70%.
Planning network budget	6,250 Mbps	6,250 Mbps	Network is rarely the first bottleneck on orchestration nodes in this model.

Table 4: VM sizing and planning headroom used in the rough-capacity math

Per-Node Building Blocks

The first step is to estimate the per-node ceiling for several workload profiles. These values are not the final product claim; they are building blocks for cluster sizing.

Profile	CPU ms/s/call	RAM MiB/call	Control-plane Mbps/call	Illustrative Behavior
Platform target thin node	0.5	2	0.02	External STT/TTS/LLM, asynchronous logging, externalized state, and no media proxying.
Routine FAQ	3.0	4	0.03	Short informational turns with modest state and telemetry.
Transactional	12.0	10	0.05	Stateful workflow with redaction, business API calls, and structured logging.
Complex regulated	25.0	18	0.08	Heavier policy checks, retries, transcript context, and richer observability.

Table 5: Illustrative per-call workload profiles used for the rough-capacity model

Table 6: Illustrative single-node capacity results from the rough math

Profile	E4 CPU Limit	E4 Memory Limit	E4 Effective	E8 Effective	First Bottleneck
Platform target thin node	5,200	11,469	5,200	10,400	CPU
Routine FAQ	867	5,734	867	1,733	CPU
Transactional	217	2,294	217	433	CPU
Complex regulated	104	1,274	104	208	CPU

The tables and figures above suggest that raw RAM is not the first limiter for an orchestration node under these assumptions. CPU saturation still arrives first. Under the platform-target thin-node profile, a Standard_E4bs_v5 node clears roughly 5,200 active calls and a Standard_E8bs_v5-equivalent node clears roughly 10,400 active calls at planning utilization. Those are building blocks for the cluster interpretation below.

Interpreting the 10,000-Call Target at Cluster Level

The more useful question for the paper is not whether one node can briefly cross the threshold, but whether the overall cluster can credibly support 10,000 active calls with maintenance and failover posture that enterprise buyers would consider reasonable.

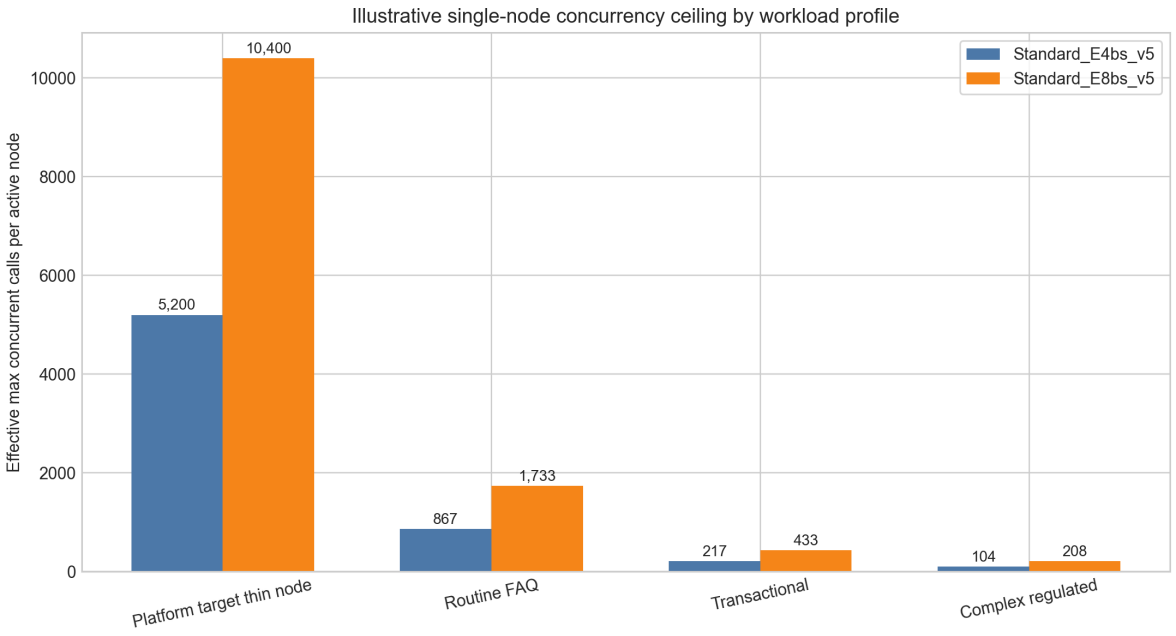

Figure 3: Illustrative per-node orchestration ceiling by workload profile.

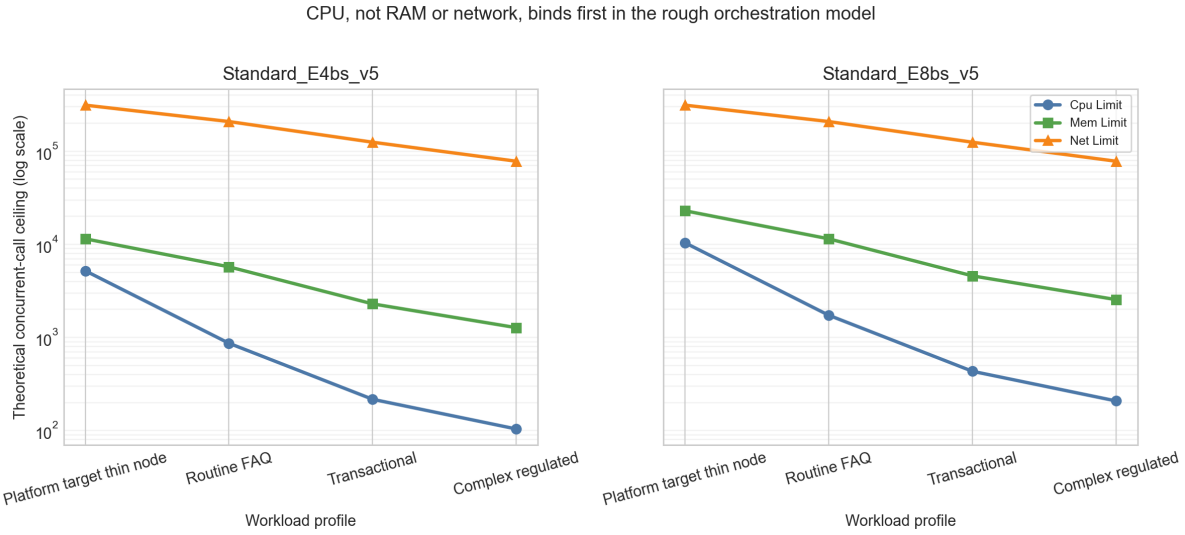

Figure 4: CPU, not RAM or control-plane network, binds first in the rough orchestration model.

Illustrative Topology	Raw Capacity	Capacity After One Node Loss	10k Cluster Target	Planning Interpretation
2 × Standard_E4bs_v5	10,400	5,200	No	Reaches the target only without spare-node tolerance.
3 × Standard_E4bs_v5	15,600	10,400	Yes	Suitable cluster-level interpretation for a 10k target with one-node-loss survival.
1 × Standard_E8bs_v5	10,400	0	No	Meets the raw threshold but does not represent a resilient cluster.
2 × Standard_E8bs_v5	20,800	10,400	Yes	Suitable cluster-level interpretation for a 10k target with one-node-loss survival.
3 × Standard_E8bs_v5	31,200	20,800 after one node loss; 10,400 after two	Yes	More conservative N+2 planning posture for maintenance plus failure tolerance.

Table 7: Illustrative cluster topologies that frame the 10,000-call target more credibly than a single-node claim

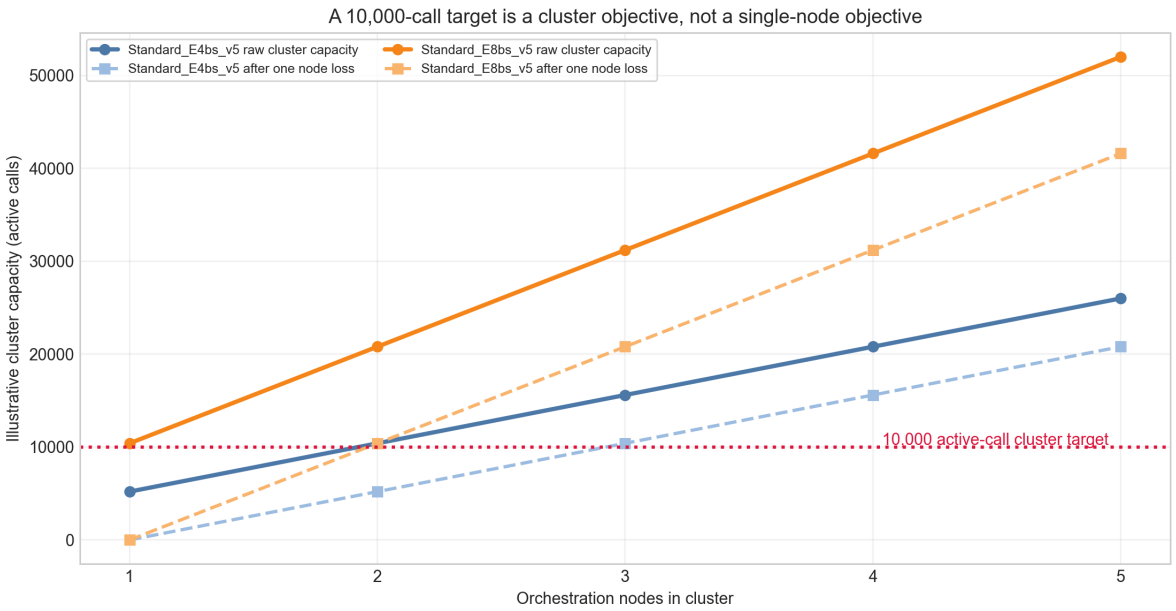

Figure 5: Illustrative raw and N+1-style surviving cluster capacity for the platform-target thin-node profile.

Under the platform-target thin-node assumptions, the paper can therefore support a 10,000-active-call cluster target. The more defensible readings are either a three-node Standard_E4bs_v5 orchestration cluster or a two-node Standard_E8bs_v5-equivalent cluster, because both preserve one-node-loss survivability while still clearing the target. For buyers that require maintenance plus unexpected-failure tolerance, the three-node Standard_E8bs_v5 option provides an illustrative N+2 posture while still surviving two node losses at the 10k target.

Sensitivity to Richer Workloads

The 10,000-call support position is highly sensitive to workload shape. Once the orchestration tier carries more stateful work per active call, active node counts rise quickly, which is why the paper should explicitly tie the 10,000 target to the thin-node platform profile.

Target Concurrent Calls	Platform Target	Routine FAQ	Transactional	Complex Regulated
2,000	1	3	10	20
5,000	1	6	24	49
10,000	2	12	47	97

Table 8: Illustrative active Standard_E4bs_v5 nodes required at planning utilization

For the platform-target thin-node profile, 10,000 concurrent calls imply roughly 2 active E4bs_v5 nodes or 1 active E8bs_v5-equivalent node before adding spare capacity. That is why the cluster-level topologies above are more credible than a single-node product claim. For transactional and regulated profiles, however, horizontal scaling remains essential and active node counts rise quickly.

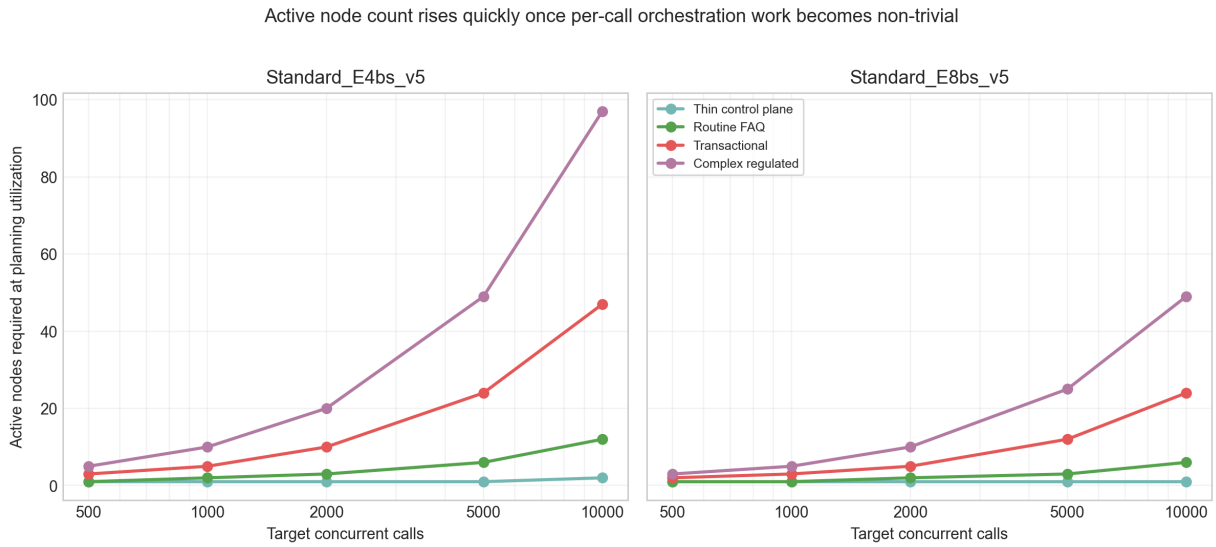

Figure 6: Illustrative active-node count required at planning utilization.

Target Concurrent Calls	Platform Target	Routine FAQ	Transactional	Complex Regulated
2,000	1	2	5	10
5,000	1	3	12	25
10,000	1	6	24	49

Table 9: Illustrative active Standard_E8bs_v5 nodes required at planning utilization

Constraints That Commonly Hit First

The exact order depends on architecture, but the following constraints commonly become material as concurrency rises.

Table 10: Common scaling constraints for high-concurrency voice AI workloads

Constraint	Why It Hits	Early Warning Signal	Common Response
Speech-to-text concurrency	One streaming STT session is typically needed per active call.	Admission failures, delayed transcription, or quota warnings before node memory is exhausted.	Distribute across speech resources or regions and obtain quota approval before testing.
Text-to-speech throughput	TTS demand scales with bot-turn density rather than call count alone.	Increased synthesis latency during busy windows and prompt playback backlog.	Reduce prompt verbosity, cache static prompts, and spread traffic across voice resources.
LLM quota and model throughput	Token demand grows with prompt size, tool use, and turn frequency.	Rising model latency, throttling, or deployment-level quota exhaustion.	Split traffic across deployments, regions, and model pools; compress prompts.

Constraint	Why It Hits	Early Warning Signal	Common Response
Orchestration CPU	Every call adds steady session-management work on the active node.	CPU saturation, event-loop lag, queue growth, and rising p95 turn latency.	Scale out on concurrent sessions, CPU, and queue depth rather than CPU alone.
Enterprise APIs	Business systems often tolerate lower sustained RPS than the AI stack.	Timeout growth, retry storms, circuit-breaker trips, and transaction backlog.	Cache aggressively, shed load, and separate read-heavy from write-heavy workflows.
Memory and transcript buffers	Long calls and verbose logging increase per-call state over time.	Gradual RSS growth, GC pressure, and container or process recycling.	Externalize state, trim transcript windows, and bound in-memory buffers.
Logging and analytics	Synchronous writes can become part of the hot path under load.	Increased turn latency during peak telemetry volume and delayed queryability.	Buffer, batch, and decouple analytics pipelines from call handling.
Human escalation queues	Degraded automation often drives more transfers at the worst possible time.	Queue delays, failed transfers, and higher caller abandonment during handoff.	Test degraded transfer paths and keep spare agent-routing capacity.

External Dependency Demand

Cluster-level orchestration math is only one part of the capacity picture. To illustrate how quickly external limits become first-class design constraints, Figure 7 and Table 11 use a balanced planning workload of 6 bot responses per minute, 800 LLM tokens per response, and 2 business API calls per minute per active call. Azure Speech publishes default limits of 100 concurrent real-time speech-to-text requests per resource and 200 TPS for real-time text-to-speech by default, with published higher adjustable ceilings for some Standard-resource cases. Azure OpenAI quota is allocated per subscription, region, and model or deployment type.[7][8]

Table 11: Illustrative dependency demand at common concurrency tiers

Concurrent Calls	STT Sessions	STT Default-Resource Equivalents	TTS TPS	LLM TPM (Millions)	API RPS
500	500	5	50	2.4	16.7
2,000	2,000	20	200	9.6	66.7
5,000	5,000	50	500	24.0	166.7
10,000	10,000	100	1,000	48.0	333.3

Taken together, the rough math supports a 10,000-call cluster target under the stated thin-node assumptions, but it remains a multi-resource and multi-quota design problem even when orchestra-

[7] Azure Speech quotas and limits
[8] Azure OpenAI quota guidance

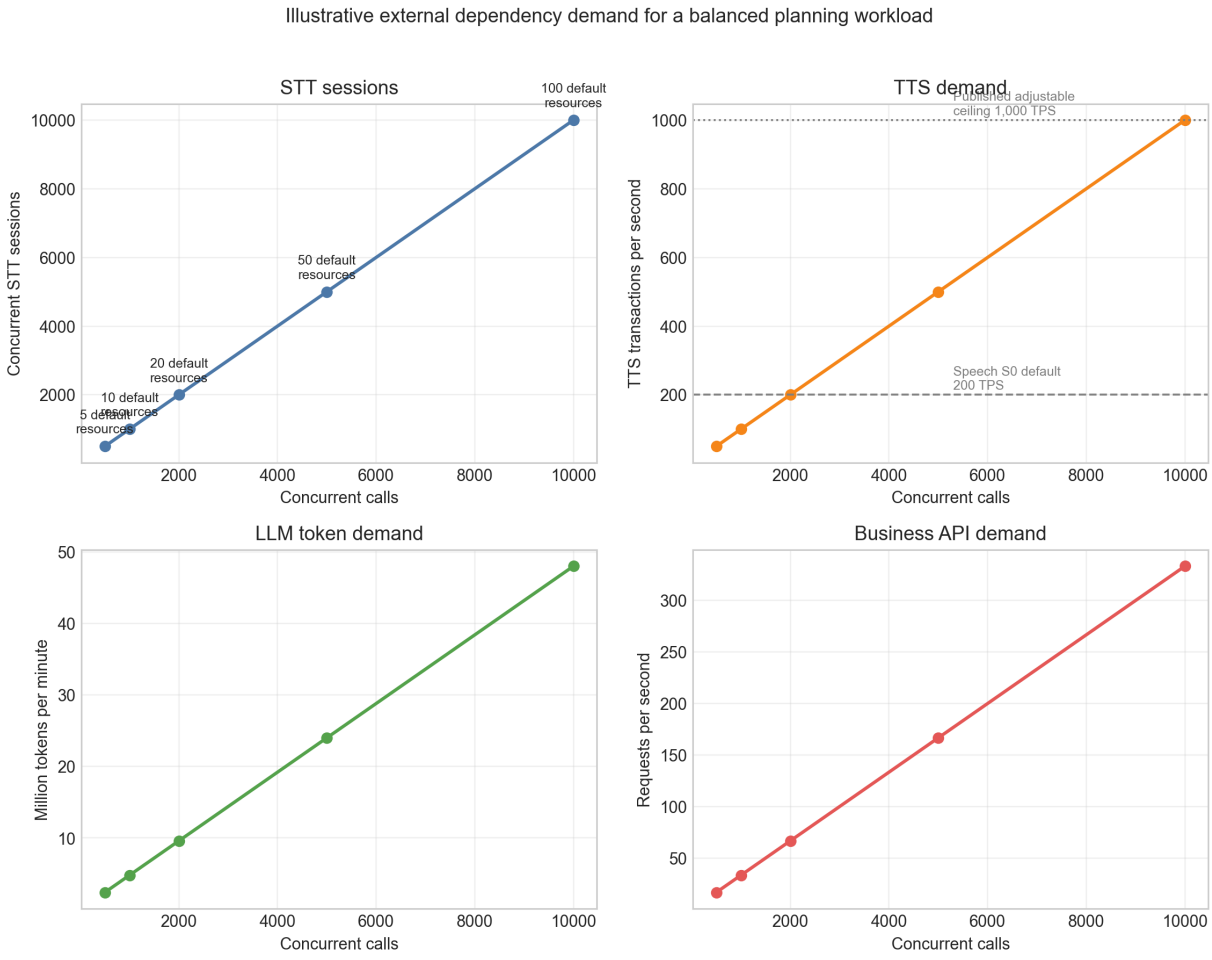

Figure 7: Illustrative external dependency demand under a balanced planning workload.

tion CPU math clears the mark. Speech, TTS, LLM, API, logging, and resilience constraints still make cluster design, quota planning, and measured validation essential.

11 Azure Scaling Considerations

Table 12 consolidates selected public Azure guidance that commonly influences concurrency outcomes. Azure autoscale documentation notes that scaling can be driven by schedules, metric thresholds, or predictive models, while Well-Architected reliability guidance recommends designing explicitly for redundancy, self-healing, and recovery.[9][10][11]

Table 12: Public Azure guidance relevant to scaling and resilience

Design Area	**Public Guidance**	**Operational Benchmark Check**	**Why It Matters**
Autoscaling	Supported autoscale patterns include metric rules, schedules, and predictive autoscale for suitable workloads.	Verify scale-out trigger timing, cooldown behavior, and time to stable latency after each scale event.	Late scaling can produce audible caller delay before infrastructure catches up.
Load balancing	Scale sets distribute traffic through load balancers, and health evaluation determines when instances should receive traffic.	Measure whether new instances receive sessions only after they are fully ready for speech and orchestration traffic.	Premature routing can inflate turn errors during bursts.
Zone redundancy	Availability-zone strategy should be chosen explicitly based on business reliability targets and tradeoffs.	Run at least one scenario with zonal impairment or partial capacity loss.	Capacity claims are stronger when they survive a partial-failure condition.
Private networking	Private endpoints and controlled service-to-service paths can change connectivity and latency characteristics.	Execute benchmark runs with the same network path and policy controls planned for production.	A public-path benchmark can overstate achievable performance in regulated deployments.
Monitoring	Workspace-based Azure Monitor resources can be designed for stronger resilience characteristics in supported regions.	Confirm that telemetry ingestion, alerting, and queries remain usable during load and fault conditions.	Production teams need observability during incidents, not only during normal operations.

Autoscaling

Autoscaling policies should be based on meaningful service indicators such as concurrent sessions, queue depth, CPU, memory, and dependency saturation rather than CPU alone. Scale-out timing

[9] Azure Monitor autoscale settings
[10] VM Scale Sets autoscale overview
[11] Azure regions and availability zones guidance

should be measured against ramp patterns to confirm the system reaches new steady states before user experience degrades.

Load Balancing

Session distribution should avoid hot spots and respect any state affinity requirements. The benchmark should verify connection distribution, health probe behavior, and the effect of instance replacement during active traffic.

Zone Redundancy

If zone redundancy is required, benchmark scenarios should validate operation across zones and controlled impairment of one zone. This helps confirm that concurrency targets remain achievable during partial infrastructure loss.

Private Networking

For regulated environments, private endpoints, network segmentation, egress controls, and secure service-to-service paths can introduce operational constraints and latency. These controls should remain enabled during benchmark execution if they will exist in production.

Monitoring

Azure-native and platform telemetry should provide enough resolution to correlate caller experience with system health. Monitoring design should support rapid distinction between telephony, speech, orchestration, model, API, and logging bottlenecks.

12 Healthcare and PHI Considerations

Figure 8 summarizes a practical control flow for regulated voice workloads where sensitive content may appear in transcripts, prompts, and audit trails.

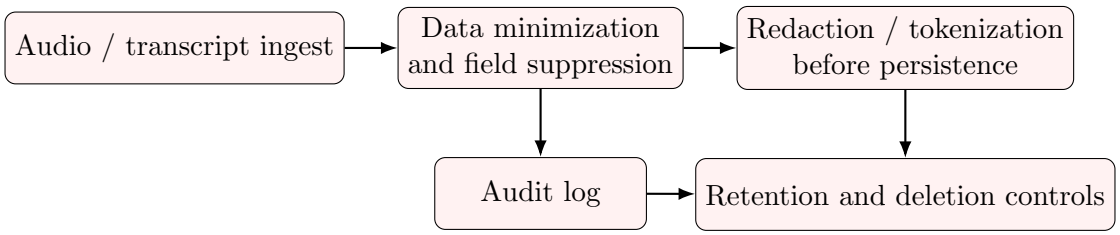

Figure 8: Illustrative PHI-aware logging and retention flow.

The HIPAA Security Rule requires regulated entities to protect the confidentiality, integrity, and availability of electronic protected health information and to implement technical safeguards such as audit controls, integrity controls, authentication, and transmission security.[12]

Table 13: HIPAA technical safeguard themes relevant to benchmark evidence

Safeguard Theme	**Public Requirement Summary**	**Benchmark Evidence to Collect**	**Source**
Confidentiality / availability / integrity	HHS states that regulated entities must ensure the confidentiality, integrity, and availability of ePHI they create, receive, maintain, or transmit.	Demonstrate that load conditions do not disable redaction, corrupt transcripts, or prevent authorized operational access.	HHS summary
Audit controls	HHS states that regulated entities must record and examine activity in information systems that contain or use ePHI.	Capture turn-level and admin-level audit events, access traces, and incident review records during peak load.	HHS summary
Integrity controls	HHS states that policies and procedures must ensure ePHI is not improperly altered or destroyed.	Validate transcript persistence, checksum or immutability patterns, and controlled retry behavior under faults.	HHS summary
Authentication	HHS states that entities must verify that a person seeking access to ePHI is who they claim to be.	Confirm operator, admin, and API identity controls remain enforced during concurrent test execution.	HHS summary
Transmission security	HHS states that technical security measures must guard against unauthorized access to ePHI transmitted over a network.	Verify encrypted transport, private-path design where required, and failure behavior when secure dependencies are unavailable.	HHS summary
Business associate agreement posture	HIPAA-regulated deployments commonly require appropriate business-associate obligations before vendors create, receive, maintain, or transmit ePHI on behalf of covered entities.	Confirm BAA status, subcontractor flow-down obligations, data-location commitments, and benchmark-artifact handling before regulated test execution.	HHS summary
Disposition / retention	HHS guidance on disposal requires policies for final disposition of ePHI and media reuse protections.	Show retention timers, deletion workflows, and evidence that benchmark artifacts follow approved disposal rules.	HHS disposal FAQ

Additional public references for Table 13 include HHS disposal guidance for PHI and ePHI media reuse.[13][14]

[12] HHS summary of the HIPAA Security Rule
[13] HHS PHI disposal FAQ
[14] HHS ePHI media reuse FAQ

Data Minimization

Only the minimum data needed to support the benchmark should be used in test transcripts, prompts, recordings, and logs. De-identified or synthetic data is preferred unless specifically approved otherwise.

Redaction

PHI and other sensitive fields should be redacted or tokenized in logs, transcripts, and analytics exports according to the intended production control design. Benchmarking should measure the latency and throughput effect of these controls while confirming they do not fail open under load.

Audit Logging

Audit logs should capture access, administrative changes, escalation events, and material workflow decisions. Audit logging must be tamper-evident and retained according to enterprise policy.

Retention Controls

Retention schedules for recordings, transcripts, prompts, and operational logs should be documented before testing. Any test data containing regulated information should be subject to the same deletion and retention controls planned for production.

13 Production Readiness Checklist

The following checklist can be used as a sign-off aid for architecture review boards, security teams, and operational stakeholders.

- Concurrency target and workload profile are documented and approved.
- Azure regional design, zone strategy, and dependency quotas are confirmed.
- BAA, subcontractor, data-location, and regulated-test-data handling requirements are confirmed where healthcare or ePHI workflows are in scope.
- Speech, LLM, and API throughput limits are validated for the target call volume.
- Telephony/CCaaS integration limits and failover behavior are documented.
- Autoscaling triggers, minimums, maximums, and warm capacity are defined.
- Timeouts, retries, circuit breakers, and fallback prompts are implemented.
- Escalation to human agents is tested under nominal and degraded conditions.

- Observability captures call, turn, dependency, and transfer-level metrics.
- Redaction, retention, and audit controls are enabled and tested under load.
- Failure-mode tests are executed and recovery behavior is documented.
- Formal benchmark evidence is archived with scenario definitions and raw measurements.
- Cost per concurrent call and cost per call-minute are estimated from the same workload profile used for latency and reliability evidence.
- Ownership for production monitoring, incident response, and capacity review is assigned.

14 Appendix: Sample Benchmark Results Table Template

Table 14 provides a sample format for reporting measured benchmark data. Replace the sample values below with actual observed values from controlled benchmark execution.

Table 14: Sample benchmark results table template

Scenario	**Target Calls**	**Observed Peak**	**Avg / p95 Turn Latency**	**Error Rate**	**Escalation Rate**	**Notes**
Baseline steady-state	Measured during test	Measured during test	Measured during test	Measured during test	Measured during test	Record environment, codec, model, and API mix.
Target concurrency	Approved target tier	Measured during test	Measured during test	Measured during test	Measured during test	Record if speech, LLM, or API quota changes were required.
Peak burst	Defined burst tier	Measured during test	Measured during test	Measured during test	Measured during test	Record recovery time and any admission-control behavior.
API degraded	Same as target tier	Measured during test	Measured during test	Measured during test	Measured during test	Record fallback prompts, timeout behavior, and circuit-breaker response.
Escalation stress	Scenario-specific	Measured during test	Measured during test	Measured during test	Measured during test	Record transfer completion, queue delay, and transcript handoff quality.
Zone or instance fault	Target tier under fault	Measured during test	Measured during test	Measured during test	Measured during test	Record graceful degradation, failover, and service-restoration observations.

For more detailed reporting, Commerce.AI may add companion tables for dependency-level metrics, autoscaling timelines, quota utilization, and security-control verification.