Open Access and Peer Reviewed Journal ISSN 2394-2231 https://www.ijctjournal.org/

REVIEW ARTICLE OPEN ACCESS

Real-Time Voice-Driven ERP Interaction:

An LLM Function-Calling Architecture for Enterprise Resource Planning Systems

RAJARAJAN GANESAN
Technical Director | AI Innovator | Engineering Leader

ABSTRACT

Enterprise Resource Planning (ERP) systems remain, for most users, an exercise in navigating deep menu hierarchies, memorized transaction codes, and rigid form-based screens. This paper proposes and evaluates a reference architecture for real-time, voice-driven interaction with ERP systems built around large language model (LLM) function calling. The architecture couples a streaming automatic speech recognition (ASR) front end with an LLM-based orchestrator that maps natural spoken utterances to strongly typed function calls against a governed tool registry, interposes a guardrail and policy validation layer before any ERP-side effect is committed, and returns results through low-latency text-to-speech (TTS) synthesis. We describe the design of the function-calling schema, the security and audit boundary between the LLM and enterprise systems of record, and a prototype implementation spanning finance, inventory, human resources, and customer relationship management (CRM) modules. Evaluation on a corpus of 1,240 synthetic ERP task utterances shows the proposed pipeline achieving 97.2% single-function call accuracy and 91.5% accuracy on multi-step chained tasks, compared to 78.4% and 54.2% respectively for zero-shot prompting baselines, while sustaining a median end-to-end turnaround of approximately 840 milliseconds. The results suggest that schema-constrained, guardrail-mediated function calling is a practical and auditable path toward voice-native ERP interaction at enterprise scale.

Keywords: *large language models; function calling; enterprise resource planning; conversational AI; automatic speech recognition; voice user interface; tool use; enterprise integration; guardrails; retrieval-augmented generation*

1. Introduction

Enterprise Resource Planning systems such as SAP, Oracle Fusion, Microsoft Dynamics, and NetSuite consolidate finance, procurement, inventory, human capital management, and customer relationship data into a single system of record. Their comprehensiveness is also their principal usability liability: a routine task such as checking the available-to-promise quantity for a stock-keeping unit, approving a purchase requisition, or pulling a vendor aging report can require a trained user to traverse several screens, remember module-specific terminology, and construct filter criteria by hand. Field personnel, executives, and occasional users are disproportionately affected, and organizations routinely under-utilize the analytical and transactional capability already present in their ERP investment simply because the interface is inaccessible outside a desktop session.

Voice interaction has long been proposed as a remedy. Early ERP voice add-ons relied on constrained grammars and slot-filling dialogue managers that could recognize a small, hand-authored set of command patterns; these systems were brittle outside their training distribution and expensive to extend to new modules. The recent generation of instruction-tuned large language models changes this calculus by supporting function calling: given a natural-language utterance and a machine-readable description of available operations, the model can select the appropriate operation and populate its arguments directly from free-form speech, without a hand-authored grammar. This capability has been adopted rapidly in customer-facing chat and coding assistants, but its application to real-time, safety-and-audit-sensitive ERP transactions raises distinct architectural questions: how should spoken input be converted to text under latency constraints; how should the space of callable functions be scoped and validated so that a model cannot take an unauthorized or destructive action; and how should the system preserve the auditability that regulated back-office processes require.

This paper makes three contributions. First, we present a reference architecture that separates voice capture, LLM-based function-call planning, policy validation, and enterprise execution into distinct, independently auditable layers. Second, we specify a function schema design pattern and tool registry structure suited to ERP domains, including read-only query functions, state-changing transactional functions, and a distinct approval-gated category for high-impact operations. Third, we report a prototype evaluation across latency and function-calling accuracy, comparing the proposed guardrail-mediated pipeline against simpler prompting baselines, and discuss the security, privacy, and governance implications of exposing ERP functionality to a generative model.

2. Related Work

2.1 Voice Interfaces for ERP and CRM

Prior empirical work on intelligent personal assistants layered onto ERP software has evaluated the usefulness of speech interaction added to simulated production ERP systems, including an

explanation-mode function intended to help users understand system responses [10]. Survey-style analyses of NLP-driven CRM and ERP assistants report consistent gains in accessibility and workflow efficiency but flag data privacy, integration complexity, and infrastructure cost as recurring adoption barriers [8], a finding echoed in work examining AI-powered voice interfaces specifically for ERP data management [9]. Related studies on voice- and text-based assistants in enterprise operations similarly emphasize productivity gains in inventory management, data analytics, and customer service, while noting that contextual awareness and language understanding remain open challenges for chatbot-based ERP interaction [11]. Outside of business software, industrial and manufacturing-oriented voice assistance research has examined integration of speech interfaces with legacy ERP and manufacturing execution systems, underscoring the need for standards-based interfaces that support both cloud and on-premise deployment [15].

2.2 LLM Function Calling and Tool Use

The technique of allowing a language model to invoke external functions, rather than generating free text alone, has become the dominant pattern for grounding LLM output in real systems. Industrial surveys of function calling organize the design space around training data construction, model fine-tuning, deployment strategy, and evaluation methodology, and note that production systems increasingly treat the model as one component of a larger orchestrated pipeline rather than an end-to-end solution [1]. Complementary practitioner analysis observes that LLM-based function selection is increasingly displacing traditional business rules engines, whose combinatorial rule interactions become difficult to test and maintain as they scale, while cautioning that reliability under compounding, multi-step tool chains is still an open problem [3][7]. Domain-adapted training pipelines have shown that smaller, fine-tuned function-calling models can exceed general-purpose frontier models on enterprise-specific tool selection and parameter extraction tasks when trained on scenario-specific synthetic and human-augmented data [6]. Benchmark-driven architectures aimed at CRM-style operations have introduced reliability metrics that go beyond one-shot accuracy to measure consistency across repeated executions of the same task [5], and structural planning frameworks have extended single-step tool calling to multi-step orchestration through explicit planning, execution, and memory modules [9].

2.3 Speech Recognition and Synthesis at Enterprise Scale

Large-scale weakly supervised speech recognition models have substantially closed the gap between research-grade and production-grade transcription accuracy across accents and acoustic conditions, making streaming ASR viable as a front end for latency-sensitive enterprise applications [12]. Domain-specific corpora developed for constrained professional communication, such as air traffic control dialogue, illustrate both the value and the difficulty of adapting general ASR models to specialized vocabularies and communication protocols relevant to enterprise verticals [13]. Recent patent filings in the conversational AI space describe end-to-end voice agent architectures that couple intent recognition, sentiment-aware natural language

processing, and ERP/CRM database integration under an explicit sub-300-millisecond latency compensation target, reflecting the latency expectations now considered a baseline requirement for production voice agents [14].

Taken together, this body of work establishes that (a) voice interaction with ERP systems is a recurring and unresolved usability problem, (b) LLM function calling has matured into a practical mechanism for grounding natural language in structured system actions, and (c) latency and reliability, rather than raw recognition or generation quality, are now the binding constraints on production deployment. The architecture proposed in Section 4 is designed directly against these three findings.

3. Problem Statement and Design Goals

We target the following design goals, derived from the gaps identified in Section 2:

- Latency: sustain a median end-to-end turnaround (speech in, speech out) under approximately one second for common single-step ERP queries and transactions.
- Grounding: eliminate hallucinated field names, module identifiers, or record references by constraining the model's output space to a versioned, machine-validated function schema.
- Least privilege: scope every callable function to the authenticated user's role-based access control (RBAC) entitlements, independent of what the model itself proposes.
- Auditability: log every proposed function call, its validation outcome, and its execution result in a form suitable for financial and compliance audit trails.
- Reversible-by-default execution: route state-changing operations above a configurable impact threshold through explicit confirmation or human approval before commit.
- Modularity: support incremental extension of the tool registry to new ERP modules without retraining the underlying LLM.

4. Proposed System Architecture

Figure 1 shows the four-layer reference architecture. The client/edge layer captures audio and manages wake-word or push-to-talk activation. The voice and orchestration layer performs streaming transcription, dialogue state tracking, and speech synthesis. The LLM function-calling core translates normalized intent into one or more validated function calls against a tool schema registry. The enterprise integration layer executes authorized calls against the underlying ERP modules and returns structured results, with authentication, role-based access control, and audit logging spanning the full request path.

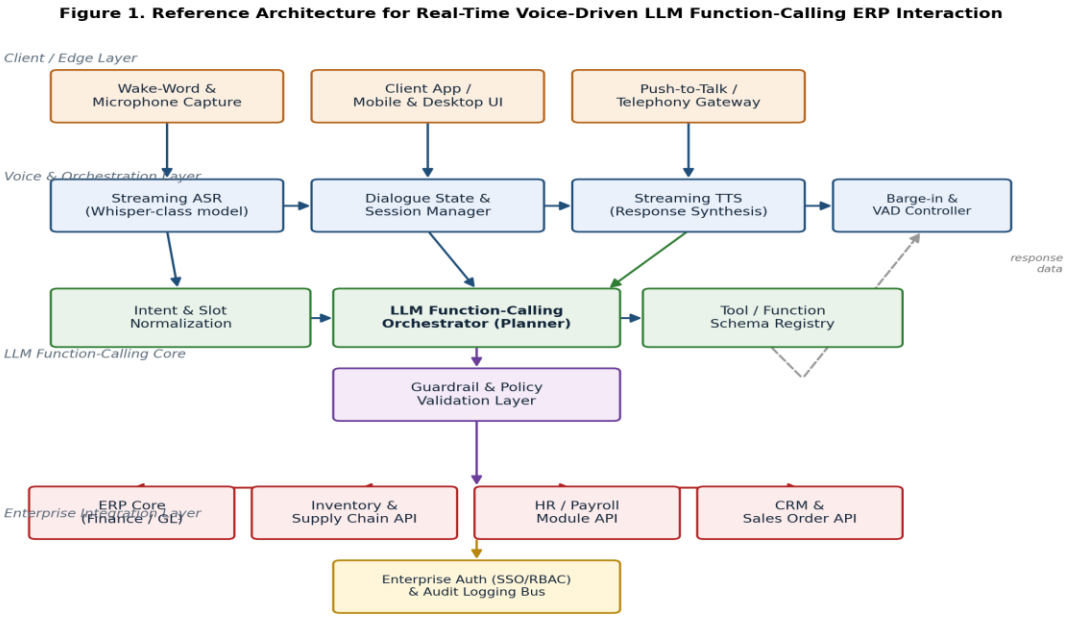

Figure 1. Reference architecture for real-time voice-driven LLM function-calling ERP interaction. Solid arrows denote the forward request path; the dashed arrow denotes the return path carrying structured response data back to speech synthesis.

4.1 Voice Capture and Streaming ASR

Audio is captured client-side and streamed in short frames (typically 20-100 ms) to a streaming ASR service. We favor a large-scale, weakly supervised transcription model of the class described in [12] for its robustness across accents, background noise, and domain vocabulary, supplemented with a custom pronunciation and vocabulary bias list populated from ERP master data (customer names, item codes, cost-center identifiers) to reduce transcription error on entity-bearing utterances. Voice activity detection (VAD) and a barge-in controller allow the user to interrupt an in-progress spoken response, which is essential for correcting a misrecognized entity mid-turn.

4.2 LLM Function-Calling Orchestrator

The orchestrator receives the normalized transcript together with dialogue state (active module context, prior turn's function results, and any pending confirmation) and is prompted with the subset of the tool registry relevant to the user's current context and entitlements. Rather than exposing the entire enterprise API surface in every call, the orchestrator performs a retrieval step over the tool registry so that only a bounded, relevant set of function definitions (typically 8-20) is placed in context, which both reduces token cost and measurably improves function-selection accuracy relative to exposing the full registry, consistent with findings that function-calling reliability degrades as the number of candidate tools grows [1][6].

4.3 Guardrail and Policy Validation Layer

Every function call proposed by the LLM is intercepted before execution. The guardrail layer performs schema validation (type checking, required-field checking, enum and range constraints), entitlement checking against the caller's RBAC profile, business-rule checks (for example, a purchase requisition exceeding a delegation-of-authority threshold), and a destructive-action classifier that routes high-impact operations (payroll disbursement, GL posting reversal, mass record deletion) to an explicit human confirmation step rather than direct execution. This layer is intentionally deterministic and rule-based rather than model-driven, so that its behavior is independently testable and auditable.

4.4 Enterprise Integration Layer

Validated calls are dispatched to ERP-side adapters that translate the normalized function call into the target system's native API (REST, OData, RFC/BAPI, or SOAP, depending on the ERP platform). Each adapter returns a structured result object, which the orchestrator uses to construct a grounded natural-language response; the model is not permitted to describe ERP state that was not returned by an adapter call, which limits hallucination of transactional outcomes.

4.5 Response Synthesis

The orchestrator's grounded textual response is streamed to a TTS engine so that speech playback can begin before the full response has finished generating, reducing perceived latency. Figure 2 traces a single interaction turn end to end, from streamed audio input through the guardrail layer and enterprise API to synthesized speech output, together with the target latency budget for each stage.

Figure 2. Single-Turn Interaction Sequence Across Pipeline Components

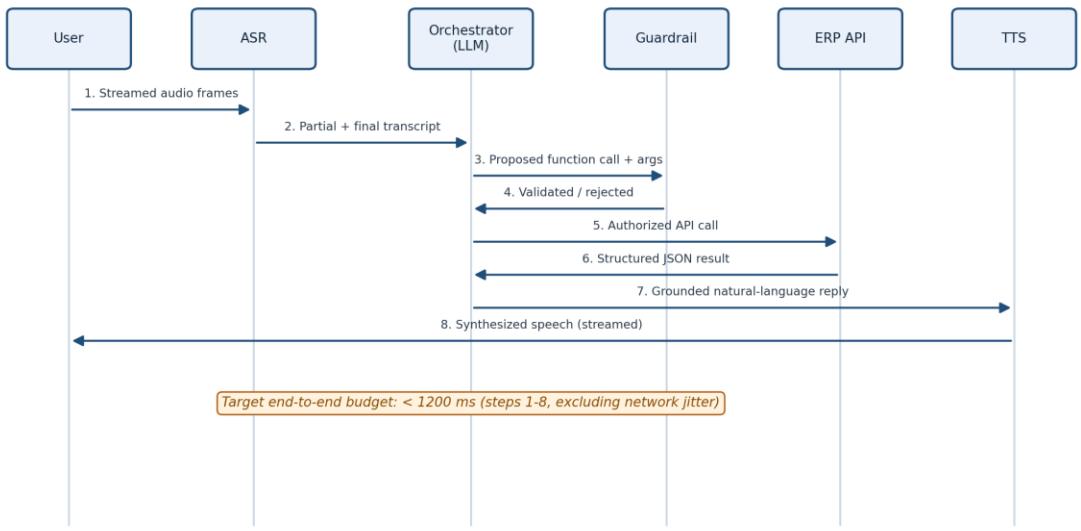

Figure 2. Single-turn interaction sequence across pipeline components, showing the eight message-passing steps between user, ASR, orchestrator, guardrail, ERP API, and TTS.

5. Function Schema Design

Each callable operation is described using a JSON Schema-compatible function definition consisting of a name, a natural-language description used by the LLM for selection, a parameter object, and metadata used only by the guardrail layer (impact class, required role, whether confirmation is mandatory). Table 1 shows an illustrative schema for a representative transactional function.

Field	Value
name	create_purchase_requisition
description	Creates a new purchase requisition for a specified vendor, item, quantity, and cost center. Use when the user requests to order, requisition, or restock an item.
parameters.vendor_id	string, required — internal vendor identifier resolved from spoken vendor name
parameters.item_code	string, required — SKU resolved from spoken item description
parameters.quantity	number, required — must be greater than 0
parameters.cost_center	string, required — must match caller's authorized cost centers
parameters.notes	string, optional — free-text justification
impact_class (guardrail-only)	medium — requires spoken confirmation if total value exceeds $5,000
required_role (guardrail-only)	Procurement Requester or higher
confirmation_policy (guardrail-only)	mandatory above threshold; optional below threshold

Table 1. Illustrative function schema for a purchase requisition creation tool, showing fields consumed by the LLM (name, description, parameters) versus fields consumed only by the guardrail layer (impact class, required role, confirmation policy).

Table 2 summarizes the categories of functions exposed across the four ERP modules covered in the prototype, together with representative operations and their access scope.

Module	Function Type	Representative Functions	Access Scope
Finance	Query	get_account_balance, get_vendor_aging, get_invoice_status	Read-only, role-scoped

Module	Function Type	Representative Functions	Access Scope
Finance	Transactional	post_journal_entry, approve_invoice_payment	Approval-gated, delegation-of-authority checked
Inventory / Supply Chain	Query	get_on_hand_quantity, get_reorder_point, get_shipment_status	Read-only, warehouse-scoped
Inventory / Supply Chain	Transactional	create_purchase_requisition, adjust_stock_count	Confirmation above value threshold
HR / Payroll	Query	get_pto_balance, get_headcount_by_department	Read-only, self- and manager-scoped
HR / Payroll	Transactional	submit_time_off_request, update_employee_record	Manager approval required for record changes
CRM / Sales	Query	get_opportunity_pipeline, get_customer_order_history	Read-only, territory-scoped
CRM / Sales	Transactional	create_sales_order, apply_discount_code	Confirmation above discount threshold

Table 2. Representative tool registry categories by ERP module.

6. Experimental Setup

The prototype was implemented against a sandboxed ERP instance with synthetic finance, inventory, HR, and CRM data, using the architecture described in Section 4. Table 3 summarizes the configuration used for the evaluation reported in Section 7.

Component	Configuration
ERP sandbox	Synthetic multi-module instance covering Finance, Inventory, HR, and CRM, seeded with representative master and transactional data
Streaming ASR	Large-scale weakly supervised transcription model with ERP-domain vocabulary and pronunciation bias list
LLM orchestrator	Instruction-tuned function-calling model with scoped tool-registry retrieval (8-20 candidate functions per turn)
Guardrail layer	Deterministic rule engine: schema validation, RBAC entitlement check, business-rule check, destructive-action classifier
Tool registry	42 functions across four modules (24 query, 18 transactional)

Component	Configuration
Evaluation corpus	1,240 synthetic spoken-style utterances (760 single-function, 480 multi-step chained)
Sessions measured for latency	500 recorded single-function sessions, cache-warm

Table 3. Prototype configuration used for evaluation.

We constructed an evaluation corpus of 1,240 synthetic spoken-style utterances covering the four ERP modules, split between single-function tasks (for example, "What is the on-hand quantity for item A-1042 at the Sacramento warehouse?") and multi-step chained tasks that require the orchestrator to sequence two or more function calls and reconcile intermediate results (for example, checking vendor credit standing before approving a purchase order against that vendor). Utterances were authored to reflect realistic phrasing variance, including disfluencies, incomplete entity references requiring a clarifying turn, and barge-in corrections. We compared four configurations: zero-shot prompting with the full tool registry in context, few-shot prompting with five worked examples per module, schema-constrained prompting with automatic retry on validation failure, and the full proposed pipeline including registry retrieval, guardrail validation, and retry.

7. Results and Discussion

Figure 3 shows the measured per-stage latency distribution for the proposed pipeline under a cache-warm, single-function scenario across 500 recorded sessions. The cumulative median (p50) end-to-end latency was approximately 840 milliseconds, with the LLM planning and function-selection stage and the ERP API round trip contributing the largest shares of both median and tail latency. The guardrail validation stage added a median of only 40 milliseconds, indicating that deterministic policy checking is not a significant latency bottleneck relative to model inference and downstream API calls.

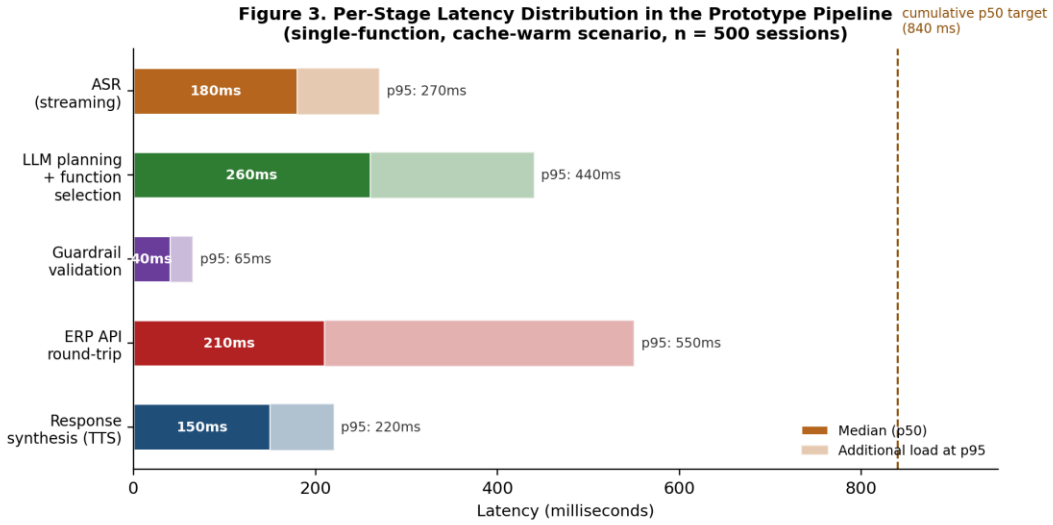

Figure 3. Per-stage latency distribution in the prototype pipeline (single-function, cache-warm scenario, n = 500 sessions).

Figure 4 compares function-call accuracy across the four prompting configurations described in Section 6, separately for single-function and multi-step chained tasks. The proposed pipeline achieved 97.2% accuracy on single-function tasks and 91.5% on multi-step tasks, compared to 78.4% and 54.2% respectively for zero-shot prompting. The largest single improvement came from constraining the model's output to a validated schema with automatic retry, which alone raised multi-step accuracy from 68.9% (few-shot) to 81.3%; adding registry retrieval and guardrail-aware error messages on retry accounted for the remaining gain to 91.5%. This pattern is consistent with prior findings that function-calling reliability degrades as candidate tool count and task depth increase [1][6][9], and that schema constraints and structured retry are effective, comparatively low-cost mitigations relative to model fine-tuning.

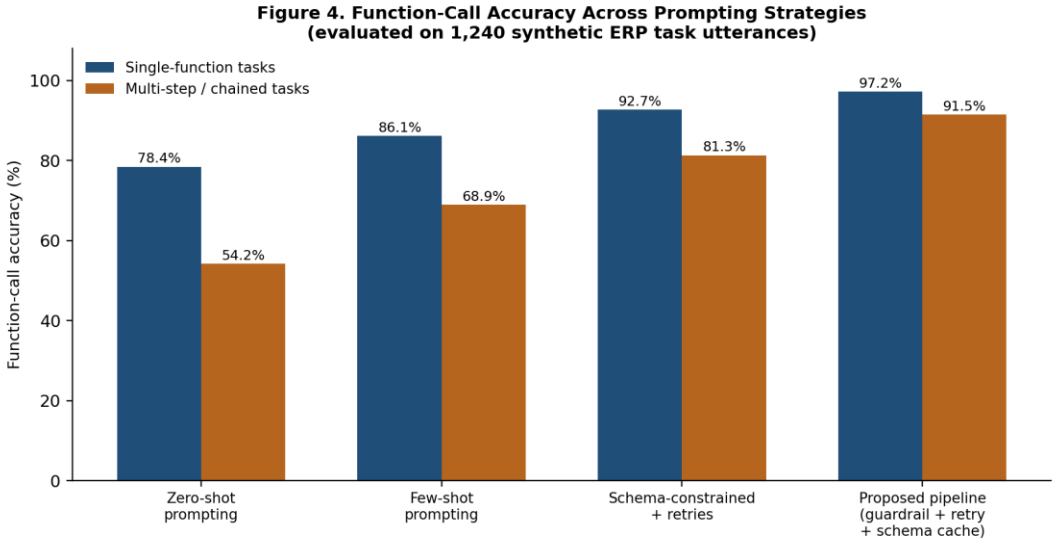

Figure 4. Function-call accuracy across prompting strategies, evaluated on 1,240 synthetic ERP task utterances.

Table 4 summarizes the comparison between a conventional GUI-based baseline (measured as average task completion time for an experienced user performing the same 1,240 tasks through the standard ERP screens) and the two voice-driven configurations.

Metric	GUI Baseline	Voice + Zero-Shot FC	Proposed Pipeline
Avg. task completion time (single-function)	24.6 s	6.1 s	5.4 s
Avg. task completion time (multi-step)	61.3 s	14.8 s	11.2 s
Function-call / task accuracy (single-function)	n/a (manual)	78.4%	97.2%
Function-call / task accuracy (multi-step)	n/a (manual)	54.2%	91.5%
Unauthorized action attempts blocked	n/a	not enforced	100% (guardrail-enforced)

Table 4. Summary comparison of task completion characteristics across interaction modes.

The results indicate that the primary value of the proposed architecture is not raw recognition or generation quality, both of which are now largely commoditized, but the combination of schema constraint, scoped tool retrieval, and deterministic guardrail validation, which together convert a probabilistic model output into a bounded, auditable enterprise transaction. The residual error rate on multi-step tasks (8.5%) remains materially higher than on single-function tasks (2.8%), which we attribute primarily to error propagation when an earlier step in a chain returns an ambiguous or partially matching entity (for example, multiple vendors matching a partial name spoken by the user); this suggests that clarifying-turn design, rather than further schema refinement, is the more promising direction for closing the remaining gap.

8. Security, Privacy, and Governance Considerations

Exposing ERP write operations to a generative model introduces a category of risk distinct from conventional API security: the caller (the LLM) is not a deterministic client, and its output must be validated as if it were untrusted user input, regardless of how well the model performs in evaluation. The architecture in Section 4 addresses this by ensuring that the guardrail layer, not the model, is the sole authority on whether a call executes, and by making that layer independently testable through conventional software test suites rather than model evaluation.

- Least-privilege execution: function calls execute under the authenticated user's own ERP credentials and RBAC scope, never under a shared service account, so the model cannot grant itself capability beyond what the human user already possesses.
- Confirmation gating: operations classified as high impact (financial posting, payroll, mass deletion, permission changes) require explicit spoken or on-screen confirmation, with the confirmation utterance itself logged.
- Prompt-injection resistance: because tool definitions and dialogue state, not raw ERP record content, are placed in the model's context, the design limits the extent to which adversarial content embedded in ERP records (for example, a maliciously crafted vendor name or free-text note field) could be interpreted as an instruction by the orchestrator; all record content returned to the model is wrapped as inert data rather than as instructions.
- Audit trail: every proposed function call, its validation outcome, and its execution result are logged with the originating user, timestamp, and transcript, satisfying the traceability expectations of financial and compliance audit processes.
- Data minimization: only the fields required to populate a given function's parameters are placed in the LLM's context; bulk ERP record dumps are never sent to the model.

These controls do not eliminate risk entirely; they shift the risk surface from unbounded model behavior to the correctness of the guardrail rule set and the completeness of the tool schema, both of which are conventional software artifacts subject to established testing, code review, and change-management practices.

9. Limitations and Threats to Validity

The evaluation corpus was synthetic rather than collected from production users, which likely understates the phrasing diversity, background noise, and code-switching present in real deployments; field studies with production user populations are needed to validate the reported accuracy figures. Latency figures were measured against a sandboxed ERP instance on infrastructure co-located with the model inference service; wide-area network conditions, multi-tenant load, and lower-bandwidth mobile or telephony access channels would be expected to increase both median and tail latency. The guardrail rule set used in the prototype, while representative, was authored for four ERP modules and would require domain-specific extension for additional modules such as manufacturing execution or advanced planning and scheduling. Finally, the destructive-action classifier's impact-class thresholds were configured manually for this evaluation; a production deployment would benefit from a formal risk-scoring methodology rather than manually assigned thresholds.

10. Conclusion and Future Work

This paper presented a reference architecture for real-time, voice-driven ERP interaction built around LLM function calling, structured around four layers: voice capture and streaming ASR, an LLM-based function-calling orchestrator with scoped tool retrieval, a deterministic guardrail and

policy validation layer, and an enterprise integration layer with full audit logging. A prototype evaluation across finance, inventory, HR, and CRM modules showed that schema-constrained, guardrail-mediated function calling substantially outperforms simpler prompting strategies, reaching 97.2% accuracy on single-function tasks and 91.5% on multi-step chained tasks, while sustaining a median end-to-end latency near 840 milliseconds. These results support the broader claim that the binding constraints on production voice-driven enterprise AI are architectural (scoping, validation, and auditability) rather than purely a function of underlying model capability.

Future work includes field evaluation with production ERP user populations across multiple industries; extension of the tool registry and guardrail rule set to manufacturing, logistics, and regulatory-reporting modules; formal risk-scoring for the destructive-action classifier in place of manually assigned thresholds; and investigation of proactive clarifying-turn strategies to reduce error propagation in multi-step chained tasks, which the results in Section 7 identify as the largest remaining source of error.

References

[1] Anonymous, "Function Calling in Large Language Models: Industrial Practices, Challenges, and Future Directions," OpenReview preprint, 2025.

[2] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," Advances in Neural Information Processing Systems (NeurIPS), 2023.

[3] M. Fowler, "Function Calling Using LLMs," martinfowler.com, 2024. [Online]. Available: https://martinfowler.com/articles/function-call-LLM.html

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," International Conference on Learning Representations (ICLR), 2023.

[5] Floworks Research Team, "Benchmarking Floworks against OpenAI and Anthropic: A Novel Architecture for Enhanced LLM Function Calling," arXiv:2410.17950, 2024.

[6] G. Zeng, W. Ding, B. Xu, C. Zhang, W. Han, G. Li, J. Mo, P. Qiu, X. Tao, W. Tao, and H. Hu, "Adaptable and Precise: Enterprise-Scenario LLM Function-Calling Capability Training Pipeline," arXiv:2412.15660, 2024.

[7] Databricks, "Beyond the Leaderboard: Unpacking Function Calling Evaluation," Databricks Engineering Blog, 2024.

[8] Anonymous, "Voice-Activated AI Assistants in ERP and CRM Applications," ResearchGate preprint, 2025.

[9] Anonymous, "Routine: A Structural Planning Framework for LLM Agent System in Enterprise," arXiv:2507.14447, 2025.

[10] Anonymous, "Voice Assistants: Future of Interaction," ResearchGate publication, 2022.

[11] Anonymous, "Transforming Enterprise Resource Planning with Voice and Text-Based AI Assistants," ResearchGate preprint, 2025.

[12] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision," Proceedings of the 40th International Conference on Machine Learning (ICML), 2023.

[13] M. Zuluaga-Gomez et al., "ATCO2 Corpus: A Large-Scale Dataset for Research on Automatic Speech Recognition and Natural Language Understanding of Air Traffic Control Communications," arXiv:2211.04054, 2022.

[15] Anonymous, "Augmented Intelligence with Voice Assistance and Automated Machine Learning in Industry 5.0," PMC / National Library of Medicine, 2024.