A Supervision-First Agent Architecture for Small-Business Workflow Automation

Yatendra Kaushik
Founder, InnoveBots.ai
Dubai, United Arab Emirates
founder@innovebots.ai

August 6, 2026

Abstract

Large language model (LLM) agents can now interpret unstructured requests and invoke external tools, which makes them a plausible substitute for the manual coordination work that dominates administrative labour in small businesses. Deployment in this setting is constrained less by model capability than by the asymmetric cost of error: a small business that sends an incorrect invoice or mishandles a customer complaint absorbs the consequence directly, with no support function to absorb it. We argue that this asymmetry should be treated as an architectural requirement rather than a deployment afterthought.

We describe a *supervision-first* architecture that separates a deterministic workflow graph from a small number of explicitly scoped agent decision points. Each agent node is constrained by a typed capability interface, a supervision policy, and a resource budget; every execution produces a replayable trace. We further describe four supervision mechanisms—shadow execution, approval gates, capability scoping, and budget limits—and characterise the class of failures each is intended to contain.

Our target deployment is small service businesses in the United Arab Emirates, a setting with properties that stress general-purpose automation: WhatsApp as the primary customer channel, code-switched Arabic–English input, and jurisdiction-specific tax handling. We specify an evaluation protocol for measuring agent reliability under deployment conditions and report its design in full. The system is in private beta; we report the architecture and protocol here and will publish empirical results on completion of the observation period.

1 Introduction

Administrative coordination consumes a substantial share of working hours in small service businesses. A salon, clinic, or small contractor typically operates a set of tools that do not communicate: a messaging channel for customers, a calendar for scheduling, an accounting package for invoicing, and a spreadsheet holding whatever the other three cannot represent. The work of moving information between these systems is performed by people, frequently by the business owner, and frequently outside working hours.

Workflow automation platforms have addressed the integration problem for over a decade. Their limitation is that they require every case to be specified in advance. A trigger fires, a condition is evaluated, an action executes. This is well suited to structured events and poorly suited to the dominant input in this setting, which is a natural-language message from a customer who does not know or care about the business's internal process.

LLM agents change the constraint. A model that can read a message, determine intent, extract relevant fields, and select an action makes the unstructured case tractable. The question is no longer whether an agent can perform the task but whether a small business can afford the consequences when it performs the task incorrectly.

We take this question to be the central design problem. In enterprise deployments, agent error is absorbed by review layers, support functions, and contractual remedy. A small business has none of these. An agent that issues an incorrect invoice creates a tax exposure the owner resolves personally. An agent that responds inappropriately to a complaint damages a relationship in a market where reputation is local and durable. The tolerable error rate is therefore not a tuning parameter but a property the architecture must guarantee.

Contributions.

(1) A characterisation of small-business workflow automation as a deployment setting, and an argument that its cost asymmetry is architecturally significant (Section 3).
(2) A *supervision-first* architecture separating deterministic workflow execution from explicitly scoped agent decision points with typed capability interfaces (Section 4).
(3) A supervision model comprising shadow execution, approval gates, capability scoping, and budget limits, with an analysis of the failure classes each contains (Section 5).
(4) An evaluation protocol for agent reliability under deployment conditions, including metrics, baselines, and study design (Section 6).

Scope. This paper describes an architecture and a planned evaluation. The system is in private beta at the time of writing and we report no empirical results. We state this explicitly rather than presenting design rationale as evidence.

2 Related Work

Agent architectures. Interleaving reasoning with tool invocation was formalised by ReAct [9], and extended by work on self-correction such as Reflexion [7]. Toolformer [6] demonstrated learned tool invocation, while Gorilla [4] and ToolLLM [5] addressed the problem of selecting correctly among large API surfaces. These establish that tool-using agents are feasible; they do not address the supervision question that arises when the tools write to systems of record owned by a third party.

Agent evaluation. AgentBench [2] and WebArena [11] evaluate agents on interactive tasks in controlled environments. τ-bench [10] is closer to our setting, evaluating agents on customer-service interactions with tool use and policy constraints. Our protocol differs in evaluating deployed traces under supervision rather than benchmark episodes, and in treating the approval-gate intervention rate as a first-class metric.

Robotic process automation. RPA systems automate structured business processes through deterministic scripting [8]. They are reliable and brittle in the same measure: they cannot accommodate input variation. Our architecture can be read as retaining the RPA execution model while replacing specific decision points with bounded agent nodes.

Human oversight. The tradeoff between automation and operator awareness is long-established in the automation literature [3], which identifies loss of situation awareness as a consequence of high automation levels. Constitutional AI [1] addresses behavioural constraint at the model level; we address it at the system level, on the assumption that model-level alignment is necessary but not sufficient when actions have external effects.

3 Problem Setting

We characterise the deployment environment along four dimensions.

Channel. The primary customer interface is a messaging application rather than a web form or telephone call. Input arrives as free text, voice notes, and forwarded images of documents. It is unstructured by default and cannot be constrained by interface design, because the interface belongs to the customer.

Language. In our target market, messages routinely code-switch between Arabic and English within a single utterance, including transliterated Arabic in Latin script. Language identification at the message level is insufficient.

Jurisdictional specificity. Invoicing requires a 5% value-added tax and a registered Tax Registration Number. The working week runs Monday to Friday. These are not localisation parameters in a system that treats them as configuration; they are correctness conditions.

Cost asymmetry. We formalise the central constraint. Let A be the set of actions available to an agent, and for $a \in A$ let $c^+(a)$ denote the value of correct execution and $c^-(a)$ the cost of incorrect execution. In enterprise deployments the ratio $c^-(a)/c^+(a)$ is moderated by review and remediation infrastructure. In our setting it is not, and for a subset of actions—those touching payment, contractual commitment, or customer communication—it is large and borne directly by the operator.

This motivates partitioning A into a set A_{auto}, for which autonomous execution is acceptable, and A_{gated}, for which it is not. The partition is a property of the business, not of the model, and must therefore be operator-configurable rather than determined at training time.

4 Architecture

4.1 Execution model

A workflow is a directed acyclic graph $G = (V, E)$ where each $v \in V$ is one of three node types:

- **Trigger nodes** initiate execution in response to an external event (an inbound message, a webhook, a schedule).
- **Deterministic nodes** perform a fixed computation: field mapping, filtering, formatting, arithmetic. Their behaviour is fully specified and reproducible.
- **Agent nodes** accept a context, invoke an LLM, and return a decision drawn from a declared output space.

The design principle is that agent nodes are a minority of V. Reasoning is introduced only where input variation makes deterministic specification impractical. This is a deliberate inversion of architectures in which an agent orchestrates the entire task and calls tools as needed: we treat the workflow as the primary artefact and the agent as a bounded component within it.

4.2 Typed capability interfaces

Each agent node v is associated with a capability interface

$$\mathcal{I}(v) = \langle \Sigma_{\text{in}}, \Sigma_{\text{out}}, T, \pi, \beta \rangle$$

where Σ_{in} is the schema of admissible context, Σ_{out} the schema of admissible output, $T \subseteq A$ the set of invocable

actions, π the supervision policy (Section 5), and β a resource budget.

Two properties follow. First, the action space of any agent node is statically known, so the set of effects a workflow can produce is determinable without executing it. Second, outputs failing to validate against Σ_{out} are rejected at the node boundary rather than propagating, which converts a class of semantic errors into recoverable structural ones.

4.3 Execution traces

Every execution emits a trace containing the trigger event, the resolved context at each node, agent inputs and outputs, actions attempted, external responses, and final state. Traces are immutable and replayable against a modified workflow definition, which supports both operator inspection and the offline evaluation described in Section 6.

5 Supervision Model

We describe four mechanisms and the failure class each addresses.

5.1 Shadow execution

Under shadow execution a workflow runs to completion with all external effects suppressed. Agent nodes produce decisions and actions are recorded as *intended* rather than performed. The operator inspects a corpus of intended actions before enabling live execution.

This addresses *specification error*: the workflow behaves as built, but as built it does not match operator intent. Such errors are invisible to model-level evaluation because the model is not at fault. Shadow execution surfaces them before they have effects, at the cost of delaying deployment by the length of the observation window.

5.2 Approval gates

An action $a \in A_{\text{gated}}$ is not executed directly. The agent prepares it, the system suspends the workflow, and an operator approves or rejects it through a notification channel. Approval decisions are recorded against the trace.

This addresses *high-cost individual errors*. It does not reduce the error rate; it relocates the decision to a party who can bear the consequence. We note the standard objection: an operator approving many low-risk actions may approve reflexively, degrading the mechanism to a formality [3]. This implies gates should be applied narrowly, and motivates the intervention-rate metric in Section 6.

5.3 Capability scoping

Each workflow authenticates to external systems with the minimum permission set its declared action space requires. Credentials are per-connection and independently revocable.

This addresses *blast radius*. It does not prevent an agent from acting incorrectly within its scope; it bounds what incorrect action can reach.

5.4 Budget limits

Each workflow carries limits on execution frequency, external calls per run, and inference spend per interval. Exceeding a limit suspends the workflow and notifies the operator.

This addresses *runaway execution*: retry loops, recursive triggering, and pathological cases where an agent repeats an ineffective action. These failures are characteristically silent and expensive.

5.5 Discussion

The four mechanisms are complementary rather than redundant, addressing specification error, individual error cost, error propagation, and aggregate resource consumption respectively. None reduces the underlying error rate of the model. We regard this as appropriate: model reliability is improved by better models and better prompts, whereas the architecture's role is to bound consequences given a reliability level it does not control.

6 Evaluation Protocol

We specify the protocol in full so that it may be assessed independently of its results.

6.1 Deployment

Participants are small service businesses in the United Arab Emirates recruited through a private beta, spanning at least three verticals, each observed for a minimum of eight weeks following a two-week shadow period. Cohort size will be reported with results.

6.2 Metrics

Task success rate. The proportion of executions reaching a correct terminal state, adjudicated against a rubric by two annotators with inter-annotator agreement reported.

Approval intervention rate. Among gated actions, the proportion rejected or edited by the operator. This is our primary measure of agent reliability on high-cost actions, and is directly observed rather than elicited.

Gate decay. Median operator response latency and rejection rate over time, testing whether approval gates degrade toward reflexive acceptance as predicted by the automation literature.

Escalation precision and recall. Whether the agent correctly identifies cases requiring human handling, measured against annotated ground truth.

Extraction accuracy. Field-level accuracy on structured extraction, stratified by input modality (text, voice transcript, image) and by language condition (English, Arabic, code-switched).

Operator time displaced. Self-reported weekly hours on automated categories, collected before deployment and at four-week intervals. We treat this as a secondary and self-report-limited measure.

6.3 Baselines

(B1) **Manual.** Pre-deployment operator handling, from the shadow period.

(B2) **Deterministic automation.** Rule-based workflows without agent nodes, covering the subset of cases expressible as rules.

(B3) **Unbounded agent.** A single agent with the full action set and no gates, evaluated offline on recorded traces. This isolates the contribution of the supervision layer from that of the language model.

Baseline (B3) is the critical comparison. Without it, results demonstrate only that an LLM is useful, not that the architecture is.

6.4 Ablations

Each supervision mechanism is disabled in turn during offline replay to estimate its contribution: gates removed (measuring incorrect actions that would have executed), scoping widened (measuring reachable-effect surface), budgets removed (measuring runaway frequency).

6.5 Threats to validity

Participants self-select into a beta and are plausibly more tolerant of error than the general population. Operators know they are observed, which may inflate approval scrutiny. Single-market deployment limits generalisation, particularly for language and tax-handling results. Adjudication rubrics are authored by the system's developers; we will publish them alongside results.

7 Limitations

A supervision-first architecture constrains consequences, not correctness. A workflow whose agent nodes are unreliable will produce unreliable proposals, and the supervision layer converts these into operator workload rather than errors—a real cost that our intervention-rate metric is intended to expose rather than obscure.

The architecture also assumes an operator willing to configure the $A_{\text{auto}}/A_{\text{gated}}$ partition. Operators who gate nothing forfeit the guarantees; operators who gate everything obtain a drafting assistant. Whether small-business operators partition sensibly in practice is an empirical question we do not yet have data to answer.

8 Conclusion

We have argued that the barrier to LLM agent deployment in small businesses is the cost asymmetry of error rather than model capability, and that this asymmetry should be addressed architecturally. The architecture confines reasoning to explicitly scoped nodes within a deterministic workflow, constrains each node by a typed capability interface, and applies four supervision mechanisms addressing distinct failure classes.

The architecture is deployed in private beta. We have specified an evaluation protocol in full and will report results, including negative results, when the observation period completes.

References

[1] Y. Bai et al. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint*, 2022.

[2] X. Liu et al. AgentBench: Evaluating LLMs as Agents. *arXiv preprint*, 2023.

[3] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(3), 2000.

[4] S. G. Patil et al. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint*, 2023.

[5] Y. Qin et al. ToolLLM: Facilitating LLMs to Master 16000+ Real-world APIs. *arXiv preprint*, 2023.

[6] T. Schick et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint*, 2023.

[7] N. Shinn et al. Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS*, 2023.

[8] W. M. P. van der Aalst, M. Bichler, and A. Heinzl. Robotic Process Automation. *Business & Information Systems Engineering*, 60(4), 2018.

[9] S. Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR*, 2023.

[10] S. Yao et al. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint*, 2024.

[11] S. Zhou et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. *ICLR*, 2024.