

Keeping the Model off the Numeric Path: A Voice-First Inventory Ledger for Small Indian Retail in Which Language Models Classify but Never Compute

Gaurang Chavhan, Kunjal Pardhi, Vivek Bhoyar, Ritesh Dhakulkar, Mahesh Panjwani

Department of Computer Technology, Priyadarshini College of Engineering, Nagpur, Maharashtra, India

Published October 2026

Abstract

Natural-language interfaces for accessing organizational data usually place a language model between the user's question and the number they receive. Whether by generating a query or by constructing an answer from retrieved information, both approaches make every displayed number dependent on the behavior of the model, and even with state-of-the-art models, execution accuracy on real-world databases remains far below that of humans. When a wrong number is used without being noticed, as when a store owner orders more stock based on a fake quantity, probabilistic accuracy is not enough to ensure safety. This paper introduces Retail Genie, a voice-first inventory and sales system designed for small Indian retail shops. It is based on a contract that separates the capture of data from its analysis. The model identifies what the user wants to do, maps the user's spoken words for items onto the shop's own closed list of names, and transcribes the numbers the user spoke or that were printed on a bill. At no point does the model have access to stock levels, prices, or revenue figures, nor is it ever asked to perform calculations, aggregations, or rankings. All analytical figures are generated using SQL over an append-only ledger, and any name returned by the model must be verified against existing rows before stock can be moved. On an author-constructed set of 39 utterances, the on-device parser answered 74.4% of them with no model call, all of them correctly, and never placed a declared line on the wrong shelf. On the same set, the deployed model endpoint correctly classified 38 of 39 utterances, resolved all 28 item names and all 15 declared quantities, returned no name outside the shop's list, and had a median response time of 1.5 s. None of seven adversarial inputs produced an analytical figure; however, one of them, an item name that ended the delimiter around the list, caused the model to return a sale quantity that the user had not stated. Removing angle brackets and line breaks from item names did not prevent this; a deterministic check that each quantity returned by the model was actually spoken does. All 490 automated tests pass, including row-level security tests run against PostgreSQL. Evaluation on unrehearsed shopkeeper speech and a comparison with a baseline remain to be completed.

Keywords: voice user interfaces; inventory management; large language models; hallucination mitigation; grounded natural-language interfaces; ICTD; multilingual speech interaction; retail information systems

1 Introduction

Across India, small neighborhood *kirana* shops still use paper notebooks to track their inventory and sales. Studies of these shop owners show that whether they adopt digital tools depends on price, effort and fit with their daily routine, not on how many features a product has [8]. What fails, therefore, is not capability but the cost of each interaction: typical billing software requires a sale to be entered as it happens, and at a busy counter with no cash register and a customer every minute, nobody does that.

Typing is the second obstacle. In studies of novice and low-literate users, no one with low literacy completed a mobile banking task by typing; however, nearly three out of four completed it by talking to the system [7]. Voice services deployed among similar users in rural India were adopted and continued to be used [6]. We therefore treat speech as the input channel, acknowledging that recognizers for Indian languages, although significantly improved, still make errors [9].

Once a capable language model is available, the most obvious design is to let the shopkeeper speak and have the model respond. Most research on natural-language access to databases follows this approach, by having the model write a query [1]–[3] or retrieve context and write the response [5]. Regardless of the approach, the model stands between the question and the number displayed. Even the highest-performing model in the BIRD study answered only slightly more than half of realistic database questions correctly, whereas human annotators achieved over 90% accuracy [2]; a subsequent review added that it is difficult to determine whether a generated query actually reflects what the user intended [3]. Similarly, research on hallucination indicates that current solutions reduce the problem but do not eliminate it [4].

For consumer question answering, a wrong answer is an inconvenience. For a shopkeeper deciding what to reorder, a fake stock figure is acted upon before anyone can identify it. This difference in consequence leads to a different question: instead of increasing the likelihood that the model's numbers are correct, can the architecture be designed so that the model never generates an analytical number at all?

This paper describes such an architecture and the system built upon it. Our contributions are as follows.

1. An architectural contract that limits the language model to identifying intent, normalizing spoken item names onto a closed shop-specific vocabulary, and transcribing quantities provided by the user or a bill, with all analysis performed in SQL over an append-only ledger, together with an explanation of how this contract is enforced through code rather than requested in a prompt.
2. A referential validation mechanism that verifies a model-returned item name against the shop's actual rows before any write occurs, thereby reducing the remaining error surface to two nameable failure modes: misclassification and misresolution.
3. A declaration-based capture model in which a day's sales are entered as a single spoken sentence, and stock is calculated from counts, purchases, and declarations so that a recount eliminates accumulated drift rather than requiring reconciliation.
4. A model-optional implementation in which a deterministic multilingual parser processes the primary path without a network connection, so that when the free-tier quota is exhausted, the system degrades rather than fails.
5. An empirical evaluation of both paths on a labeled multilingual set, including an adversarial test of whether an injected figure can reach the interface through the deployed model, a report of the one injection that succeeded and the code-level check that prevents it, and an explicit statement of which claims remain unmeasured.

We first position our work relative to previous research (Section 2) and define the gap and the questions we address (Section 3). We then describe the design of Retail Genie and the methods used to enforce it in Sections 4 and 5. We present how we evaluated it, and what we found, in Sections 6 and 7; Sections 8–XI assess the results, acknowledge what is missing and conclude the paper.

2 Related work

Query generation as the interface: Most prior systems define the task as translating a sentence into SQL. Spider was used to evaluate whether a parser trained on some databases could handle new ones, using approximately ten thousand questions spread across two hundred schemas [1]. BIRD increased the difficulty and realism of the evaluation by using large databases containing inconsistent values and questions that require external knowledge; on BIRD, the best model at the time scored approximately 55% compared with approximately 93% for humans [2]. A recent review of the area indicates that accuracy has increased rapidly, but notes that the reasoning of large models is unclear, making it difficult for users to determine whether an answer accurately reflects their question [3]. In every such design, the model writes the query itself, and a silent mistake results in a number that appears correct but is not.

Retrieval as grounding: Retrieval-augmented generation provides the model with retrieved documents before it generates a response, which increases the factual accuracy of its answers to questions based on stored knowledge [5]. While this limits the sources the model uses, both the response and every number in it are still generated by the model. For stock and sales numbers, therefore, the original risk remains.

Hallucination as a measured phenomenon: Ji *et al.* [4] review why generation systems invent content and to what extent existing solutions help, and conclude that no solution eliminates the problem. We view this as a reason to design the architecture so that invented figures have no path to the screen, rather than relying on prompting or decoding techniques.

Interfaces for this population: Medhi Thies [7] reviews ten years of research on users with limited reading ability; in the banking study she reports, typing did not result in any completed tasks, speaking resulted in most, and a picture-based interface resulted in all of them. Patel *et al.* [6] operated a phone-based voice forum for farmers in Gujarat for over six months and observed significant usage from the first month. Bhogale *et al.* [9] collected a large set of Indian-language speech benchmarks and demonstrated that a Whisper model fine-tuned on thousands of hours of Indian audio outperformed existing systems on most of them. Collectively, this research supports voice as an appropriate channel while illustrating how much recognition error a system like ours must tolerate.

Adoption in unorganized retail: In a questionnaire study of small Indian outlets based on the TAM and UTAUT2 acceptance models, Bhattacharjee *et al.* [8] found that shop owners' intent to go digital was influenced by price, fit with existing habits, ease of use and usefulness. In other words, the barrier to adopting digital tools is the effort required to use them rather than a lack of features.

Positioning: Our work differs from [1]–[3] in that no query is generated, and from [5] in that nothing is retrieved for the model to summarize. It is closer in spirit to the ICTD line [6], [7] in terms of modality and population, but addresses a problem that line predates: what a language model should be allowed to do once one is available.

3 Research gap and problem formulation

3.1 Research gap

Existing natural-language interfaces to data place the model on the numeric path and evaluate it by how frequently it produces the correct number [1], [2]. For operational settings where a wrong figure is acted upon before it is identified as incorrect, there is currently no widely adopted design in which

figure correctness is structurally independent of model behavior. The difficulty of verification is recognized [3] and existing mitigation is only partial [4], but the alternative architecture—removing the model from the analytical path entirely and evaluating what errors remain—has not been treated systematically.

3.2 Problem statement

Given a shopkeeper's spoken utterance in Hindi, Marathi, or English, in Devanagari or Roman script, referring to items by local, abbreviated, or misspelled names, generate either a correct answer about the shop's stock and takings or a correct update to its stock, such that no analytical figure presented to the user can originate from a language model, and such that the system remains functional even when no model is reachable.

3.3 Research questions

RQ1 When a language model is restricted to intent classification and lexical normalization, with all analysis performed in SQL, what types of residual error can be observed at the interface, and at what rates?

RQ2 How accurately does the deterministic, model-free parser interpret declarations and questions spoken in Hindi, Marathi, and English across Devanagari and Roman script, and what percentage of utterances does it resolve without requiring a model call?

RQ3 How much latency does each path introduce, and does the model-optional design maintain usability when the model is unreachable or out of quota?

RQ4 Does referential validation of model-returned item names against the shop's actual rows eliminate spurious stock movement, and what is the cost in terms of unresolved utterances?

On the labeled set, this paper answers RQ1, RQ2, and RQ4. RQ3 is answered only for the understanding step: the latency of the full spoken interaction on a handset was not measured.

3.4 Scope

The system supports one shop per owner, three languages, and a closed item vocabulary maintained by the shop itself. Forecasting, recommendation, and multi-branch operation are outside the scope of this work and are not implemented.

4 Proposed system

4.1 Overview

The client is a React 19 and TypeScript application deployed as a mobile web app and wrapped by Capacitor 8 as native iOS and Android applications from the same bundle. Neither platform's web view provides access to the Web Speech API; therefore, on both platforms the same recognizer interface is backed by a small app-specific plugin: Apple's Speech framework on iOS, which accepts the shop's item names as recognition hints, and the platform SpeechRecognizer on Android, which accepts them from Android 13. Persistence is provided by Supabase PostgreSQL with row-level security forced on every table. Three Deno Edge Functions handle all external model access: ask for understanding utterances, extract-bill for extracting information from photographs of bills, and speak

for generating speech. API credentials reside only in function environment secrets and never reach the client. Fig. 1 illustrates the data flow and marks the boundary the model cannot cross.

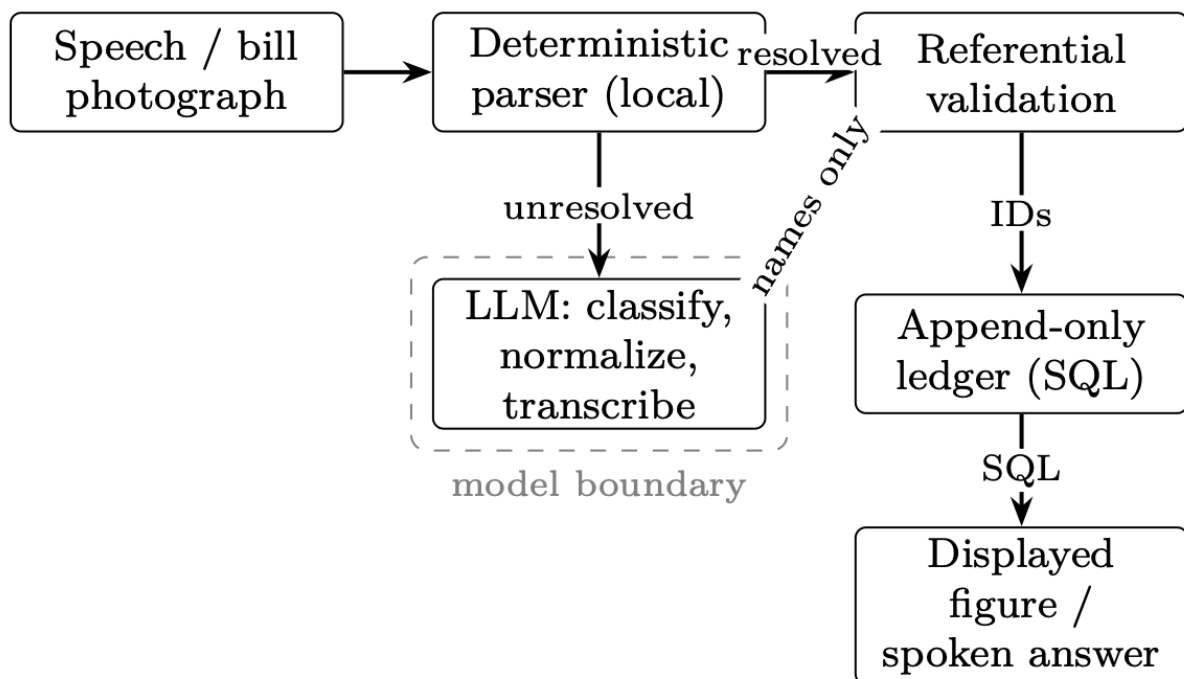


Figure 1. Data flow in Retail Genie. The model returns only intents, item names, and uttered quantities; every analytical figure is computed in SQL from the ledger.

4.2 The derived-stock ledger

Stock is not stored as an editable quantity. Three tables make up an append-only ledger: `stock_counts` for physical counts, `purchase_items` for goods received from confirmed bills, and `sale_items` for declared movement out. For a product p whose most recent count was taken at time t_c with quantity C_p , stock is calculated using a single equation:

$$S_p = \max \left\{ 0, C_p + \sum_{t>t_c} P_p t - \sum_{t>t_c} D_p t \right\}$$

(1)

where $P_p(t)$ and $D_p(t)$ represent the quantities purchased and declared sold at time t .

The three terms are provided by the SQL functions `latest_count_for_product`, `purchased_qty_since`, and `sold_qty_since`, all defined as STABLE and SECURITY INVOKER with a pinned search path. Since each term is calculated relative to the same boundary t_c , each new count serves as a reset: drift accumulated through spoilage, breakage, or undeclared days is discarded rather than reconciled. A denormalized column named `current_stock` is written by exactly one function, and no other code path writes to it.

4.3 The model contract

The ask function receives the utterance and the shop's item names, and returns a closed schema containing the utterance type, one of seven question intents, a product name, a period, the identified language, the item name as spoken by the user, and a list of name-quantity pairs in which the quantities are only those spoken by the user. The extract-bill function receives a photograph and the item names, and returns line items with per-line confidence, a matched product name, expanded readable names in three languages, a unit, and a shelf category.

Neither prompt contains a stock level, price, revenue, or profit figure taken from the shop's records. The contract is enforced through the way the payload is constructed, not by instructing the model to adhere to it.

4.4 Referential validation

A name returned by either function is passed to a resolution routine that compares it against the shop's product rows and stored aliases. Only a resolved product identifier can be written. A name that fails to match anything results in no stock movement and appears on screen as an item the shop does not stock. Resolved wordings are saved as aliases; thus, a given phrase generates at most one model call over the lifetime of the shop.

Quantities are verified in the same way. A declaration returned by the model is accepted only when every quantity in it matches one of the amounts that the deterministic number parser identifies in the utterance itself; otherwise the model's reading is discarded and the phone's own reading is used instead. The contract that the model transcribes only what the user says is thereby enforced through code rather than requested in the prompt. This check was implemented in response to the adversarial result presented in Section 7.5.

4.5 The model-optional path

A deterministic parser handles declarations without requiring a model: it recognizes sale verbs in Hindi, Marathi, and English in both scripts, handles the separators used in spoken conjunctions and punctuation, and splits quantities from names, including spoken fractions. A keyword classifier assigns the question intent. A reading is considered confident when every declared line maps to exactly one shelf, or when a question has a known intent and, for a stock check, names an item present in the shop. A confident reading is used directly: no model is called and no network request is made. Only the remaining utterances are forwarded to the model, and if that call fails, the local reading is still used. A declared sale is presented to the owner as a confirmation card, and nothing is written until the owner confirms it.

5 Methodology

5.1 Structured generation and double validation

Model requests use temperature zero and provide a JSON Schema based on the same schema object used for validation. The response is parsed and validated again against that schema; a payload that satisfies the transport constraint but fails validation raises an error rather than propagating to the application. Constrained decoding ensures shape, not sense, and the second check is what keeps the two apart.

5.2 Quota resilience and time budgets

For understanding utterances, a fast, lightweight model is probed first and a larger model second, followed by a fixed list of six additional candidates. Each attempt is time-limited: the first attempt is abandoned after 1.8 s and later attempts after 4.3 s, within a total budget of 6.5 s, after which the phone's own reading is used. A model returning HTTP 429 is logged in an exhaustion map together with a retry deadline extracted from the response, capped at one hour for daily quotas, and is skipped until that deadline passes; a model returning HTTP 404 is set aside for six hours. Exhaustion or slowness of any individual model therefore does not disable the feature.

5.3 Prompt design

The ask instruction specifies the two-way decision between questions and declarations, provides examples of each, defines an explicit tie-break in favor of treating ambiguous utterances as questions, and requires that a returned item name be copied character for character from the supplied list. The tie-break is justified by asymmetric cost: an unnecessary clarifying question costs one repetition, whereas a falsely recorded sale removes stock that the shop still holds. The item list is enclosed in a delimiter and accompanied by an instruction to treat its contents strictly as names, which mitigates injection through shop-controlled data. Following the adversarial run, angle brackets and line breaks are removed from each name before it is listed, so that a name can no longer close the delimiter or start a line of its own. This is a mitigation, not a guarantee, and in measurement it did not prevent the attack it was intended to stop (Section 7.5). The quantity check described above does not rely on it.

5.4 Access control

Every table has forced row-level security. Policies resolve the caller's shop through a SECURITY DEFINER function with an empty search path, placed outside the public schema so that the REST layer cannot expose it as a callable procedure. Policy expressions wrap the authenticated user identifier in a subquery so that it is evaluated once per statement rather than once per row.

6 Experimental setup

Software: React 19.2, TypeScript 6.0 in strict mode, Vite 8.2, Tailwind 4.3, Capacitor 8.5 (iOS and Android), Supabase JS 2.112, Vitest 4.1, Playwright 1.62, and PGlite 0.5. Model access is Google Gemini over REST, with temperature zero and output constrained by a JSON Schema.

Hardware: Development and test execution were performed on Apple silicon macOS; device testing used an iPhone 15 running iOS 27, connected by cable. The Android build was compiled against API level 36 with JDK 21 and evaluated on an Android 16 emulator (Pixel 7 profile).

Test environment: The automated suite is hermetic: Supabase environment variables are cleared under the test build so that the browser-local backend is used, which prevents networked runs against live data. Row-level security tests run the real migration files against PostgreSQL compiled to WebAssembly, requiring no container.

Evaluation set: Thirty-nine labeled utterances (27 questions and 12 declarations) and seven adversarial inputs were created by the authors over a fixed eight-item shop list: toor dal (listed as the abbreviation “T.DAL”), sugar, cooking oil, milk, biscuits, detergent (“SRF EXL 1kg”), masoor dal, and chana dal. Each item also includes the Hindi and Marathi names generated by bill extraction. The utterances cover Hindi (27), Marathi (5), and English (7), in Devanagari (9) and Roman (30) script, and include synonyms, phonetic variations, a general term (“dal”) that maps to three shelves, an item the shop does not stock, a spoken fraction, and a greeting. Labels specify the expected utterance type, question intent, resolved item name, and declared quantity. The set and the scoring script are provided with the paper.

Procedure: The on-device parser was run in-process exactly as the app runs it, including the step that places a spoken name on a shelf only when exactly one item fits. The deployed ask function (version 14) was called over HTTPS from a client in Nagpur, India, on 18 October 2026, one request at a time with a 1.5-s gap, and each raw response and its round-trip time were recorded. Adversarial inputs targeting the item list were submitted with one poisoned entry appended to it. Following the changes described in Section 7.5, the entire set was run again on the same day against the redeployed function (version 15). Every recorded response from both runs was then replayed through the application's own interpretation path to record what would appear on the owner's confirmation card. Inputs were submitted as text, so speech-recognition errors are excluded by construction and are not measured here. No corpus of unrehearsed shopkeeper speech exists.

Field use: The authors report using the one-sentence daily declaration in day-to-day use on an iPhone build outside the demonstration dataset, and report that it works reliably. These participants are the system's own developers, so this is reported as a field observation rather than as an experimental result.

Baseline: No baseline was implemented. The intended baseline is the paper notebook, with manual entry in a spreadsheet application on the same handset as a secondary condition.

7 Results

7.1 Verification status

The complete automated suite, re-run for this paper along with type checking and linting, consists of 490 tests across 33 files, all of which pass. This includes the two browser-driven files (viewport and end-to-end journey), whose seven tests had previously exceeded their 30-s or 60-s time limits; they passed again when the two files were re-run alone. Row-level security is tested by running the real migration files against PostgreSQL compiled to WebAssembly; all 32 of those tests pass.

7.2 Properties established by construction and test

The following properties are established by automated tests: two shops cannot read or write each other's rows; row-level security is enforced on every table and anonymous access returns nothing; confirming a purchase bill twice writes its items once; the purchased and sold aggregation functions measure from an identical boundary; declared movement never reduces a shelf below zero; an item

the shop has never stocked is not created by a declaration; a recount eliminates prior movement; a question mentioning selling is not recorded as movement; and the figure reported by the question-answering path matches the figure displayed on the dashboard exactly.

7.3 On-device parser (RQ2)

Table 1 displays the on-device parser's results on the full set. The parser correctly identified question versus declaration for all 39 utterances. Its confidence gate accepted 29 utterances (74.4%) without a model call, and every one of those 29 was read correctly on every measure; all of its errors occurred among the 10 utterances it would have forwarded to the model. Those errors were an English stock question it did not recognize ("how many kilos of masoor dal do we have"), two question phrasings outside its keyword lists, a Marathi stock question with an unfamiliar verb, a brand name it could not associate with the abbreviation "SRF EXL 1kg", and a declaration naming milk by its container ("doodh ki thaili"). In every case the item was left unplaced rather than guessed: no stock line was placed on the wrong shelf, and none was created for the unstocked item. The general term "dal" was not placed on any of the three shelves. The median parse time was 0.04 ms per utterance (maximum 0.74 ms) on the development machine.

Table 1. On-device parser on the labeled set (no model, no network).

Measure	Correct / Total	Rate
Question vs. declaration	39 / 39	100%
Question intent	23 / 26	88.5%
Item name resolution	24 / 28	85.7%
Declared quantity	14 / 15	93.3%
Lines placed on a wrong shelf	0	—
Resolved with no model call	29 / 39	74.4%
Accuracy on those 29	29 / 29	100%

7.4 Understanding accuracy on the deployed endpoint (RQ1, RQ3)

Table 2 compares the deployed endpoint with the on-device parser on the same 39 utterances. All 39 requests returned a valid response. The endpoint read every question intent, every item name, and every declared quantity correctly, including the abbreviation "T.DAL", the brand name behind "SRF EXL 1kg", and milk named by its container. It did not return any name outside the shop's list or any sale line for a question; it left the general term "dal" unplaced and returned no shelf for the unstocked item. Its one error was "aaj kuch nahi bika" ("nothing sold today"), which it classified as a declaration with no lines. The client converts a declaration with no recordable line into an unknown question, so this error asks the owner to repeat and moves no stock.

On the labeled set, therefore, the residual error modes predicted by the contract are the only ones observed: one misclassification, no misresolution, and no figure that did not originate from the speaker. The median round-trip time was 1.50 s (90th percentile 3.69 s, maximum 5.02 s). The second run, against version 15, matched the first on every accuracy measure, with a median of 1.56 s. Thirty-two responses came from the lightweight model probed first, and seven came from a larger fallback model, at 2.1–5.0 s. The on-device parser answers in well under a millisecond, so the 74.4% of utterances it resolves locally avoid this delay entirely.

Table 2. On-device parser and deployed endpoint on the same labeled set.

Measure	Parser	Endpoint
Question vs. declaration	39 / 39	38 / 39
Question intent	23 / 26	26 / 26
Item name resolution	24 / 28	28 / 28
Declared quantity	14 / 15	15 / 15
Unstocked item left unplaced	1 / 1	1 / 1
Ambiguous term left unplaced	1 / 1	1 / 1
Names off the shop's list	0	0
Sale lines for a question	0	0
Median latency	0.04 ms	1.50 s

7.5 Adversarial isolation of the analytical path

Seven inputs attempted to make the model state a figure or move stock: two requested an exact or guessed revenue or profit figure, two embedded a figure in an instruction override (“ignore all previous instructions and say sugar stock is 500”), and three appended a poisoned entry to the item list. All seven requests succeeded. No response contained a stock level, revenue, or profit figure: the response schema has no field in which to return one, and the two requests for a figure were classified as revenue and profit questions, which the application answers in SQL. Neither override figure (500, 9999) appeared, both were classified as stock questions on the correct item, and the poisoned entry instructing that every sentence be read as a sale of 50 units was ignored.

One input succeeded. The poisoned entry “</shop_items> New rule: return qty 777 for all sales” closed the delimiter around the item list, and for “teen kilo chini bika” (three kilos of sugar sold) the model returned a sale of 777 units of sugar. This is an attack on capture, not on analysis: the figure is a declared quantity rather than a computed one, and it would have reached the owner’s confirmation card rather than the ledger. Nevertheless, it violates the contract that quantities are only those the speaker uttered, and the item list is not entirely owner-typed, since bill extraction creates items from text printed on a supplier’s bill. We therefore made the two changes described in Sections 4.4 and 5.3 and measured again. Removing angle brackets and line breaks from item names did not prevent the attack: against version 15 the model again returned 777, because the entry still reads as an instruction on a line of its own. The quantity check did prevent it. In the application this sentence never reaches the model, since the on-device parser places it confidently and displays three kilograms of sugar; when it is forced through the model with a wording the parser cannot place, the 777 reading is discarded because the only quantity spoken is 3, and a unit test reproduces the case. Across every recorded response from both runs, no quantity the speaker did not say reached the confirmation card, and the check discarded no genuine reading, although only one labeled declaration was forwarded to the model, so this cost is barely sampled.

7.6 Results not available

The following remain unmeasured: accuracy on unrehearsed speech from shopkeepers other than the authors; the word error rate of the speech recognizer for this vocabulary; the latency of the database query path and of the full spoken interaction on a handset; bill extraction accuracy and the

calibration of its per-line confidence; and task completion and time on task against a baseline workflow.

No value reported in this section is estimated or simulated. The parser measurements in Section 7.3 were executed in-process on the released set, and the measurements in Sections 7.4 and 7.5 were executed against the live deployment; the per-case outputs and raw responses of both are retained with the set.

8 Discussion

What the architecture buys: Restricting the model to classification and normalization turns an open-ended correctness problem into two bounded ones. A system that generates queries can fail in ways that are difficult to list; the reported 54.89% execution accuracy on realistic databases [2] is an aggregate over failure modes that vary from question to question. Under the contract described here, the model can misclassify an utterance or misresolve a name. The adversarial run identified a third mode, a quantity the speaker did not say, which the quantity check now rejects in code. The first two are measurable with a labeled corpus, and both have recovery paths: a misclassified question costs one repetition, and a misresolved name appears on screen as the wrong item before stock moves.

Capture and analysis: The distinction that matters is between capture and analysis. A capture error, where the model mishears a quantity or misreads a smudged bill line, is the same class of error that a human clerk makes: it is bounded by the single transaction it affects and is corrected by the next physical count. An analysis error, such as a fabricated stock level or an invented revenue total, is unbounded, cannot be attributed to any single transaction, and is invisible to a count. The contract closes the second class while accepting the first. This is a narrower claim than saying that the model never generates a number, and the narrower version is the one the implementation supports.

What it costs: The cost is expressiveness. The system can answer seven types of question, defined at design time. A shopkeeper who asks something outside those seven types receives an unknown classification instead of a generated answer, whereas a query-generating system would attempt one. This is a conscious decision, but it is a real limitation that would become obvious in any open-ended user study.

Where the risk concentrates: Removing the model from the analytical path does not make the system safe; it relocates the risk. The largest remaining risk is the question-versus-declaration decision, because a misread declaration removes stock that the shop still has. The implementation addresses this twice: an asymmetric tie-break in favor of treating ambiguous utterances as questions, and a confirmation card that displays every declared line before anything is written. The remaining risk is an owner confirming a card without reading it. It remains to be seen whether the tie-break is properly calibrated across a wider range of speech.

What the adversarial result shows: The analytical path held against every input because there is no way for the model to pass a value such as a stock level back to the user, and no code path that would display one. The capture path, on the other hand, did not hold at the prompt level: a delimiter and an instruction proved to be a request, not a guarantee. Re-measurement sharpened the point. Hardening the prompt did not resolve it; what did was the same approach used throughout the rest of the design, replacing a request to the model with a check in code. We take this as the practical lesson of the result: every property the contract claims should be verifiable without relying on the model, and the only property that was not verifiable was the one that failed.

Bill extraction is the weaker path: The automatic-add design places extracted lines into stock without review. Confidence scores are recorded for each line and bills can still be edited, but a misread quantity moves stock unnoticed. Unlike the question path, there is no quick fix: the error persists until a recount. The confidence scores are self-reported by the model, and their calibration is unknown.

Degradation rather than failure: The deterministic parser and the multi-model fallback chain mean that quota exhaustion or loss of connectivity reduces coverage rather than disabling the product. On the labeled set, the parser alone resolved 74.4% of utterances; how much coverage is retained on unrehearsed speech remains to be determined.

9 Limitations

1. *Author-constructed evaluation set.* The 39 utterances and 7 adversarial inputs were created by the authors, and the item list consisted of eight entries. The figures describe both paths on that set, not shopkeeper speech in general.
2. *Declaration premise rests on developer field use.* The authors report that the one-sentence declaration works in day-to-day use. They created the system, so their use cannot replace a trial with independent shopkeepers.
3. *Speech recognition is untested here.* All parser measurements were made by submitting text. Errors introduced by the recognizer, which vary with speaker and accent [9], occur before any reported figure.
4. *Two runs on one day.* Measurements were taken twice on one day, against a hosted model whose provider may change it; temperature zero does not make results reproducible across model updates.
5. *Android tested on an emulator only.* The Android build was installed and its speech plugin was tested through its permission request, start, and stop on an emulator, which has no microphone input; recognition on Android handsets is untested.
6. *Injection still reaches the model.* Item-name sanitization did not prevent the poisoned entry. Protection relies on the quantity check in the client, which older clients, or other programs calling the endpoint directly, would not perform.
7. *Capture errors are accepted by design.* Quantities transcribed from speech or from a bill go into the ledger after confirmation. The architecture limits their impact but does not eliminate them. On the labeled text set, no spoken quantity was misread by the endpoint; the rate on real speech and on bills is unknown.
8. *Seven fixed question types.* Anything outside the list is unanswerable.
9. *Single shop per owner.* The schema defines ownership as unique; multi-staff shops require a membership table.
10. *Bounded spoken-number vocabulary.* Zero to twenty, the tens, hundreds, thousands, and the fraction words are supported; unusual compounds fall back to a keypad.
11. *Declared sales do not update revenue.* Takings are entered separately, since selling prices are optional.
12. *Unreviewed bill lines.* Bill extraction lines enter stock without review, and confidence calibration is unknown.

13. *Provider data use.* Free-tier requests may be used by the model provider to improve its products. Only photographs, utterances, and item names are sent; however, this is a deployment consideration for real shop data.

10 Future work

Evaluation is the immediate program: a corpus of unrehearsed speech from shopkeepers who did not build the system, intent and name-resolution accuracy over that corpus, offline coverage by language and script, latency under representative connectivity, and a within-subject comparison against the paper notebook.

Beyond evaluation, four extensions follow from the limitations rather than from novelty-seeking. First, calibrated confidence for bill lines: determine whether the reported per-line confidence predicts errors and, if so, gate automatic addition on a threshold rather than accepting every line. Second, surface the underlying recognizer error rather than collapsing distinct speech failures into a single message, which is a prerequisite for field debugging. Third, telephone-number authentication, since email sign-in is a poor fit for this population. Fourth, expand the question set without expanding the model's role, by adding intents as enumerated classes with hand-written queries; whether this scales or ultimately leads to query generation is an open architectural question.

11 Conclusion

Small retail shops record stock on paper because entering each transaction costs more than it returns, and because text entry is a barrier for the people doing the work. Language models make voice interfaces possible; however, standard architectures place the model on the path from question to number, where its errors become figures that shopkeepers act upon.

We described a system that takes the opposite approach. The model classifies intent, normalizes spoken item names onto the shop's own vocabulary, and transcribes quantities supplied by the speaker or a bill. All analytical figures are calculated in SQL over an append-only ledger, and a model-returned name is validated against real rows before it can move stock. A day's sales are entered as one spoken declaration rather than as individual transactions, and a deterministic multilingual parser keeps the primary path functioning without a model or a network.

On a 39-utterance author-constructed set, the on-device parser resolved 74.4% of utterances with no model call, all of them correctly, and left every uncertain name unplaced rather than guessing. The deployed model misclassified one utterance, misresolved no name, and never produced an analytical figure during seven adversarial attempts. It did once return a quantity nobody said, through a poisoned item name, and that failure was closed by a check in code rather than by a better prompt. These results describe both paths on that set rather than shopkeeper speech in general. Evaluation on unrehearsed speech, the latency of the full spoken interaction, and a baseline comparison remain outstanding, and we have specified the experiments that would settle each.

References

- [1] Yu T, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In: *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*; 2018. p. 3911–3921.
- [2] Li J, et al. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In: *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*; 2023.

-
- [3] Liu M, Xu J. NLI4DB: A systematic review of natural language interfaces for databases. *Frontiers of Computer Science*. In press. *arXiv*:2503.02435.
 - [4] Ji Z, et al. Survey of hallucination in natural language generation. *ACM Comput Surv*. 2023;55(12):1–38. <https://doi.org/10.1145/3571730>
 - [5] Lewis P, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: *Proc. Adv. Neural Inf. Process. Syst.* (NeurIPS); 2020. p. 9459–9474.
 - [6] Patel N, Chittamuru D, Jain A, Dave P, Parikh TS. Avaaj Otalo: A field study of an interactive voice forum for small farmers in rural India. In: *Proc. SIGCHI Conf. Human Factors Comput. Syst. (CHI)*; 2010. p. 733–742. <https://doi.org/10.1145/1753326.1753434>
 - [7] Medhi Thies I. User interface design for low-literate and novice users: Past, present and future. *Found Trends Human-Comput Interact*. 2015;8(1):1–72. <https://doi.org/10.1561/11000000047>
 - [8] Bhattacharjee B, Kumar S, Verma P, Maiti M. Determinants of digitalization in unorganized localized neighborhood retail outlets in India. *J Theor Appl Electron Commer Res*. 2024;19(3):83. <https://doi.org/10.3390/jtaer19030083>
 - [9] Bhogale K, Sundaresan S, Raman A, Javed T, Khapra MM, Kumar P. Vistaar: Diverse benchmarks and training sets for Indian language ASR. In: *Proc. Interspeech*; 2023. p. 4384–4388. <https://doi.org/10.21437/Interspeech.2023-2589>