International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

SmartshopAI: An AI-Assisted Retail Management Framework for Small Shops

Om Prashant Kulawade
Jitendra Prakash Choudhary
Vitthal Anil Dikule
Department of Artificial Intelligence and Machine Learning
Zeal Polytechnic, Narhe, Pune, Maharashtra, India

ABSTRACT

Small retail shops often manage billing, stock records, and reordering manually or through disconnected tools. As a result, recorded stock can diverge from shelf conditions and replenishment decisions can be delayed. This paper proposes SmartShopAI, an integrated retail management framework combining billing, inventory control, configurable low-stock alerts, short-term sales forecasting, a controlled natural-language query interface, and image-based shelf monitoring. Each confirmed sale is treated as an atomic transaction that updates stock and checks reorder thresholds, while the analytical modules read completed sales records and produce advisory outputs. The paper contributes a layered architecture, a data model with an audit trail, safety constraints for AI-assisted querying and vision alerts, and an evaluation protocol with defined metrics. It is a design and evaluation-protocol paper: it reports no measured accuracy, user-study results, or field outcomes. Its findings concern design requirements and the evidence needed before the system can be relied on in daily retail work.

Keywords – retail management, inventory control, demand forecasting, natural-language query, computer vision, small shops

I. INTRODUCTION

Product availability on the shelf is a recurring problem in retail. A worldwide study of fast-moving consumer goods retail reported an average out-of-stock rate of roughly 8% [1], and a later analysis by its authors concluded that out-of-stocks had changed little over the following decade [2]. A missed stock update can make an item appear available when it is not on the shelf, while a late reorder loses sales and an excessive reorder ties up cash in slow-moving products.

Most of this evidence comes from large chains and manufacturers. The situation of independent shops, where billing, stock checking, and reporting are often manual or handled by separate tools, is less well documented. This paper therefore treats small-shop operation as the design context and does not assume that findings from large retailers transfer directly.

SmartShopAI addresses this setting with a single operational system. Its core loop is deliberately simple: record a sale, reduce the matching stock quantity, compare the result with a reorder threshold, and update alerts, reports, and the dashboard. Forecasting, natural-language querying, and shelf-image analysis are decision-support features built on top of this transactional base. They inform the shopkeeper but do not replace the shopkeeper's judgement.

The paper makes three contributions. First, it specifies a layered architecture in which the transactional path is separated from advisory analytics. Second, it defines a data model and integrity rules that keep stock quantities auditable and distinguish database stock from visual shelf observations. Third, it sets out safety constraints and an evaluation protocol for the three intelligent modules. The paper does not claim that forecasting, text-to-SQL, or object detection are new techniques. Its contribution lies in integrating them under explicit validity controls for a small-shop setting, and in defining how each claim should be tested. The scope is one shop, English-language questions, and shelf images from that shop.

II. BACKGROUND AND RELATED WORK

A. Demand Forecasting

Retail demand is a time series that varies by day, week, product, season, and local activity. Lim and Zohren [3] survey deep and classical forecasting methods and note that the appropriate choice depends on the amount of historical data, the forecast horizon, and the need to explain the output. For a single-shop prototype, simple baselines are therefore the natural starting point. Random forests, which average many decision trees, are a widely used non-linear regression method and can serve as a comparison model when features such as day of week, recent sales, price, and promotion status are available [4].

B. Natural-Language Access to Databases

Natural-language interfaces can reduce the effort of retrieving routine information, but translating a question into SQL is difficult. The Spider benchmark evaluates text-to-SQL systems on complex queries over databases that do not appear in the training data; the authors report that the best model at the time reached only 12.4% exact-match accuracy in the database-split setting [5]. This result motivates restricting the first version of SmartShopAI to a documented set of business questions, rather than to open-ended generation.

C. Object Detection for Shelf Observation

The YOLO family frames object detection as a single regression problem that predicts bounding boxes and class probabilities in one network evaluation [6]. The original authors note that this design makes more localisation errors than some detection systems, although it produces fewer false positives on background regions. OpenCV provides image-processing functions that are useful for preprocessing, capture, and visualisation [7]. Reflections, occlusion, packaging variation, camera angle, and poor lighting can all affect detection in a shop, so shelf monitoring should be treated as a screening tool rather than as proof that a product is absent.

D. Research Gap

Each component above is established in its own domain. The gap addressed here is the integration problem: how a small shop can combine a reliable transactional core with forecasting, language querying, and vision screening without allowing an advisory output to silently change inventory. This paper has not carried out a systematic review of commercial retail systems, so this gap is stated as a design gap that a formal review should confirm before submission.

III. PROBLEM STATEMENT AND RESEARCH QUESTIONS

The problem addressed is the lack of an affordable, integrated decision-support system for small shops in which billing, stock records, reports, and shelf checks are connected. When these are disconnected, staff reconcile information manually and may discover an unavailable item only when a customer asks for it. The system must keep a dependable record of sales and stock, and its intelligent features must be understandable and reviewable.

The work distinguishes core retail functions from optional AI-assisted features. The core functions are billing, stock updates, and low-stock alerts; they must work without the AI modules. The AI-assisted functions are forecasting, natural-language querying, and shelf monitoring; they are advisory and can be disabled without affecting the core. The paper addresses four research questions, each linked to a measure in Section VIII:

RQ1. Do the defined transaction rules keep stock quantities and sale records consistent with expected values across normal, cancelled, duplicate, return, and concurrent-sale scenarios?

RQ2. Does a tree-based forecasting model outperform simple baselines on later time periods that were not used for training?

RQ3. What proportion of supported natural-language questions return a correct query and a correct answer?

RQ4. Can shelf-image alerts identify low-availability regions with acceptable precision and recall, measured against staff-verified labels?

IV. PROPOSED SYSTEM ARCHITECTURE

SmartShopAI uses a layered architecture, shown in Fig. 1. The presentation layer contains the billing screen, inventory forms, dashboard, query box, and image-upload or camera interface. The application layer validates requests and coordinates the billing, inventory, alert, reporting, forecasting, language, and vision services. The data layer stores durable records in a relational database. Separating the layers allows the shop to keep operating when the forecasting or vision module is unavailable.

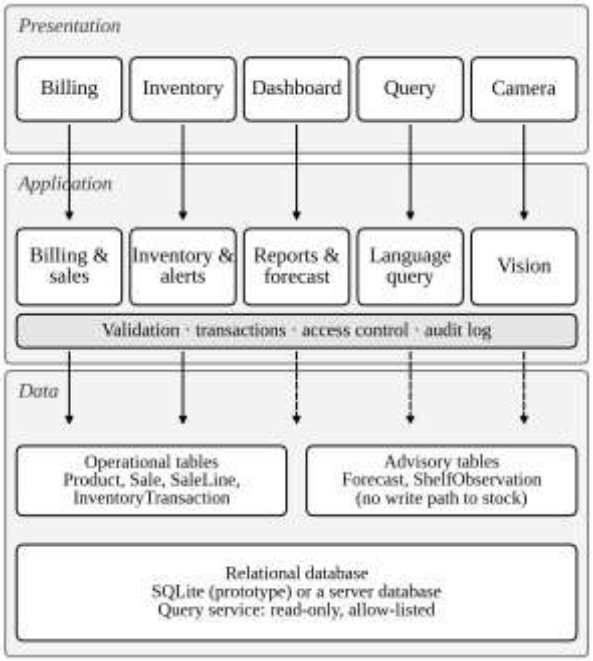


Fig. 1. Layered architecture of SmartShopAI. Solid arrows show the core transaction path; dashed arrows show read or advisory access.

The transaction path is the most important part of the system. After the cashier confirms a bill, the application writes the sale header and sale lines in one database transaction. It then decreases the relevant product quantities and checks each resulting quantity against the configured reorder level. If a product is at or below that level, an alert is created and remains visible until a user reviews it. If any step fails, the transaction is rolled back so that no partial sale is recorded. Reports and forecasts read completed sales only, so unsaved billing input cannot distort summaries.

Two kinds of information are kept separate. The stock quantity in the database is an accounting record that changes only through logged transactions. A ShelfObservation is a visual signal from the camera that suggests an area may look less full than expected. An item can be present in the store room but missing from the shelf, so the two signals are displayed side by side and neither overwrites the other. Table I summarises the expected behaviour of each component.

TABLE I
CORE COMPONENTS AND EXPECTED BEHAVIOUR

Component	Expected behaviour
Billing	Calculates the bill and saves a completed sale in one transaction.
Inventory	Maintains product quantity and reorder level; logs every change.
Alerts	Creates a visible restock prompt for review when the threshold is reached.
Forecasting	Estimates short-term demand, with uncertainty noted (advisory).
Language query	Returns a validated data answer in plain English (read-only).
Shelf monitoring	Flags a possible availability issue for staff inspection (advisory).

V. DATA MODEL AND OPERATIONAL LOGIC

Table II lists the entities of the proposed data model. The model is designed so that every change to a stock quantity can be traced to a reason, a time, and a user. The current quantity alone does not explain how it was reached, so an inventory transaction log is kept alongside it.

TABLE II
PROPOSED DATA MODEL

Entity	Key attributes	Purpose
Product	ID, name, category, price, quantity, reorder level	Stable product record
Sale / SaleLine	Time, total; product, quantity, unit price	Completed bills
InventoryTransaction	Product, change, reason, user, time	Audit trail of stock
User	Name, role, hashed password	Access control
Alert	Product, trigger time, status, reviewer	Low-stock prompts
Forecast	Product, horizon, estimate, model version	Advisory demand
ShelfObservation	Image time, region, confidence, reviewer decision	Visual signal

Several integrity rules follow from the model. A sale is not committed if it would create a negative quantity, unless the shop explicitly enables backorders. Prices and product records are validated before billing. A stock adjustment requires a user, a timestamp, a reason, and a quantity difference. Duplicate

submissions, such as a repeated click on the confirm button, should be detected with a unique sale identifier so that one bill cannot be saved twice. Staff accounts have role-based permissions, and passwords are stored only as secure hashes. These rules are design requirements; their implementation must be verified with the tests in Section VIII.

VI. INTELLIGENT DECISION-SUPPORT MODULES

The three modules below are proposed designs. None of them is claimed to be implemented or validated in this manuscript. Each module produces advisory output that a person must review.

A. Sales Forecasting

The forecasting module predicts daily or weekly sales for each product. Candidate features include lagged sales, day of week, month, category, selling price, and known promotional periods. The evaluation begins with two transparent baselines: repeating the last observed period, and a moving average over a fixed window. A random-forest regressor [4] is then trained as a non-linear comparison, using the scikit-learn implementation [8]. Training and test data are separated by time: the model is fitted on earlier periods and evaluated only on later periods, so that no future information leaks into training. The model with the simplest design that performs acceptably on the held-out periods should be preferred.

Forecast error is measured with the mean absolute error (MAE) and the root mean squared error (RMSE):

$$\mathrm{MAE} = (1/n)\,\Sigma_{t=1}^{n}\,|y_t - \hat{y}_t| \qquad (1)$$

$$\mathrm{RMSE} = \sqrt{((1/n)\,\Sigma_{t=1}^{n}\,(y_t - \hat{y}_t)^2)} \qquad (2)$$

where y_t is the observed sales in period t, $\hat{y}_t$ is the forecast, and n is the number of held-out periods. Results are reported separately for fast-moving and slow-moving products, because a single average can hide poor performance on the slow movers. Forecasts are shown as estimates together with the sales history they are based on. Where a product has little history, has not sold recently, or shows unusual variation, the interface should indicate low confidence. The reorder decision remains with the shopkeeper, who knows supplier lead times and events that the data may not capture.

B. Natural-Language Query

The language module answers a limited set of questions, such as the total number of items sold on a given day, the best-selling products in a date range, and products at or below their reorder level. A rule-based interpreter with fixed intents is suitable for the first version, because each intent, date expression, and product reference can be tested directly. Each intent maps to a parameterised SQL template, and the answer is returned in concise English together with the date range used, so that the user can see what was asked.

The safety controls are essential. Generated queries are restricted to read-only operations. They are checked against an allow-list of tables and fields, and they run under a database account that cannot modify operational data. Where a question cannot be mapped to an intent with confidence, for example because a product name or date range is unclear, the system asks for clarification rather than guessing. A more flexible language model may be considered only after the constrained version has been evaluated, and it would remain inside the same controls.

C. Shelf Monitoring

The vision module accepts a shelf image and estimates whether monitored regions appear less full than expected. It is proposed as a pipeline with four stages: image-quality checks, detection of products or empty space, comparison with a reference layout for each shelf region, and generation of a reviewable observation. A candidate detector from the YOLO family [6] may be used, with OpenCV [7] for preprocessing. The detector choice should be made on the shop's own images, not on the claims of a generic benchmark.

Training and test images must represent the actual shop. They should cover different lighting, viewing angles, package types, partly blocked products, and both full and low-availability shelves. Annotation rules must be written down before labelling starts. Frames taken from the same shelf at nearly the same time must not be split between training and test sets, since this would inflate the results. A confidence threshold is chosen on a validation set, and every alert is shown to staff, who confirm whether it reflects a shelf gap, a misplaced product, or a false detection caused by glare or occlusion. The module never reduces inventory automatically, because an image cannot reveal every operational cause of an apparently empty shelf.

VII. IMPLEMENTATION METHODOLOGY

[Authors to confirm: the current implementation status (planned, partially implemented, or implemented), the software versions used, and whether any code is publicly available. If no implementation exists yet, this section describes a planned method only.]

A practical prototype can use Python for application logic, SQLite for a single-shop database, and a lightweight interface built with HTML, CSS, JavaScript, or Streamlit. The schema is designed before the interface, so that billing and inventory updates can be tested as atomic operations. A later multi-user version may use MySQL or another server database, role-based access, and a scheduled backup process.

Development proceeds in four stages. Stage 1 delivers product setup, billing, inventory updates, and low-stock alerts, because these create the reliable data needed by later stages. Stage 2 adds reports and the baseline forecasts. Stage 3 adds the constrained query interface. Stage 4 prototypes shelf monitoring on a limited set of shelf images. To keep the work reproducible, each stage should record the schema version, the software versions, the random seeds, the exact train and test date boundaries, and the test cases with their expected outputs. Completed functions must be re-tested after every later change.

VIII. EVALUATION PROTOCOL

The evaluation has two parts: the operational correctness of the transactional core, which must be established first, and the usefulness of the AI-assisted features. Table III lists each measure, the data required, and its current status. **No measured results are reported in this manuscript.** A status changes from "Not run" only when the corresponding test has actually been carried out.

TABLE III
EVALUATION MEASURES AND STATUS

Question	Measure	Data required	Status
RQ1 Transactions	Agreement of stock and sale records with expected values	Scripted scenarios, incl. duplicate and concurrent sales	Not run
RQ2 Forecasts	MAE and RMSE against baselines on later periods	Daily sales history for the shop	Not run
RQ3 Queries	Share of correct queries and correct answers	Question set with reference answers	Not run

Question	Measure	Data required	Status
RQ4 Shelf alerts	Precision, recall, F1 at a validation-set threshold	Labelled shelf images, split by time	Not run
Usability	Task time; clarity of alerts; support for restocking (rating scale)	Sessions with shop staff	Not run

For RQ1, each scenario is run against a fixed starting database, and the resulting sale record, stock quantity, and alert state are compared with the expected values. Scenarios include a normal bill, a cancelled bill, a duplicate submission, a return, a stock adjustment, a threshold crossing, and two simultaneous attempts to sell the final unit. RQ2 uses the time-separated protocol in Section VI-A and reports MAE and RMSE for each baseline and the candidate model, separately for fast and slow movers. Because the paper compares models on one shop's data, the results will describe that shop; they should not be generalised without further data.

For RQ3, a set of realistic shop questions is written in advance, with reference SQL and reference answers. Reviewers check both the executed query and the final English answer for correctness, completeness, and clarity. For RQ4, precision, recall, and F1 score are computed at the intersection-over-union threshold fixed in advance, using staff-verified labels from a held-out time period. The usability study measures task completion time and asks staff to rate the clarity of alerts and whether the dashboard supports restocking decisions. All results, including failure cases, must be reported, not only the favourable ones.

IX. LIMITATIONS, PRIVACY, AND SECURITY

The main limitations are structural. Forecasts depend on representative sales history; new products, sudden local events, and products with irregular demand cannot be predicted reliably from past sales. The query interface supports only the questions it was designed for and may misread vague wording. Vision quality can degrade with glare, shadows, occlusion, changing packaging, or a camera moved from its original position. Accurate software cannot compensate for missing deliveries, incorrect product setup, or sales that were never recorded. Results from a single shop may not transfer to other shops with different layouts, product ranges, or customers.

Privacy and security also require attention. The system should store only information needed for shop operations. Access should be controlled through individual accounts and role-based permissions, and passwords must be stored as secure hashes. Backups must be protected. Customer contact details and payment-related information should be minimised and handled according to the applicable data-protection rules in the shop's jurisdiction. Shelf images can capture people incidentally. Cameras should point at shelves, images should be retained only as long as review requires, and staff should be informed about the monitoring. Before any deployment, the shop should confirm that its use of cameras complies with local law.

Throughout, the AI outputs are treated as recommendations. Human users remain responsible for operational decisions, and no advisory output may change inventory records without a logged human action.

X. CONCLUSION AND FUTURE WORK

This paper presents SmartShopAI, a layered framework for small-shop retail that integrates billing, inventory control, alerts, and three AI advisory modules. Its core contribution is its architectural design, which features a transactional core with explicit integrity rules, isolated visual-versus-database stock tracking, and AI safety constraints.

No performance metrics are claimed; the expected benefits—improved stock accuracy, automated reorder alerts, and reduced routine effort—are hypotheses to be tested via the empirical protocol in Section VIII. Next steps include implementing the core, collecting a retail dataset, and evaluating all modules against Table III metrics.

.

ACKNOWLEDGMENT

The authors thank Prof. Sneha Shinde for guidance throughout the development of this work.

REFERENCES

[1] T. W. Gruen, D. S. Corsten, and S. Bharadwaj, "Retail out-of-stocks: A worldwide examination of extent, causes and consumer responses," Grocery Manufacturers of America, Washington, DC, USA, 2002.

[2] D. Corsten and T. Gruen, "Desperately seeking shelf availability: An examination of the extent, the causes, and the efforts to address retail out-of-stocks," *Int. J. Retail Distrib. Manag.*, vol. 31, no. 12, 2003, doi: 10.1108/09590550310507731.

[3] B. Lim and S. Zohren, "Time-series forecasting with deep learning: A survey," *Philos. Trans. R. Soc. A*, vol. 379, no. 2194, Art. no. 20200209, 2021, doi: 10.1098/rsta.2020.0209.

[4] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[5] T. Yu *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proc. EMNLP*, 2018, pp. 3911–3921, doi: 10.18653/v1/D18-1425.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[7] G. Bradski, "The OpenCV library," *Dr. Dobb's J. Softw. Tools*, 2000.

[8] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.