

DataPilot: A Self-Hosted AI-Powered Data Analytics Platform with Self-Learning Decision Intelligence

Fevinkumar Dineshbhai Raval¹

*1Department of Computer Science & Engineering,
Parul Institute of Engineering & Technology, Parul University, Vadodara, Gujarat, India
Email: fevinraval88@gmail.com*

Abstract - The explosive growth of enterprise data has created a strong demand for analytics platforms that are simultaneously powerful, affordable, and deployable on-premise. Commercial tools such as Tableau (\$70+/user/month) and Microsoft Power BI are constrained by high subscription costs and cloud dependency, while existing open-source alternatives such as Apache Superset and Metabase lack integrated machine learning and conversational analytics. This paper presents DataPilot, a self-hosted, full-stack, AI-powered data analytics platform that combines dashboarding, automated machine learning, natural language querying with multi-turn conversation memory, and a novel self-learning decision intelligence engine within a single application. The system is implemented as a three-service microservice architecture: a Node.js backend with Express and MongoDB; a React 19 frontend with TypeScript; and a Python Flask ML microservice using scikit-learn and XGBoost. A streaming ETL pipeline ingests files up to 5 GB with bounded memory, and a two-level persistent aggregation cache (in-process LRU combined with MongoDB TTL collection) achieves up to 2,060× query performance improvement on a 500,000-row benchmark. Evaluation on five real-world datasets demonstrates a best regression R^2 of 0.974 using Random Forest, 92% intent-classification accuracy on a 50-query benchmark, and cached response times below 50 milliseconds.

Key Words: Data Analytics, Machine Learning, Natural Language Processing, MongoDB, Self-Learning AI, Dashboard Builder, ETL Pipeline

1. INTRODUCTION

The past decade has witnessed unprecedented growth in the volume, velocity, and variety of data generated across every industrial sector. Industry reports estimate that global data creation now exceeds 2.5 quintillion bytes per day. This explosion has created a strategic challenge for organisations: deriving actionable insight from data at a cost and operational complexity that small and medium-sized businesses can realistically support.

The market for data analytics tools has grown substantially in response, but it has evolved in a manner that disproportionately favours large enterprises with significant IT budgets and dedicated data-science teams. Commercial platforms dominate the mid-to-large enterprise segment, while open-source alternatives partially address affordability but sacrifice key capabilities — particularly integrated machine learning, natural language querying with conversation context, and AI-driven decision support.

This paper presents DataPilot, a self-hosted, full-stack analytics platform that combines all five capabilities —

dashboarding, ML training, NL querying, AI decision support, and time-series forecasting — within a single application that can be deployed on-premise with zero subscription cost and complete data sovereignty. The platform was designed and implemented during a three-month internship at Ultron Technologies Pvt. Ltd., VV Nagar, Gujarat, India, from January 2026 to April 2026.

The key contributions of this work are: (i) a three-service microservice architecture that isolates ML computation in a dedicated Python service; (ii) a streaming ETL pipeline capable of ingesting 5 GB files on modest hardware; (iii) a novel two-level aggregation cache that delivers up to 2,060× query speedup; (iv) an automated 7-algorithm ML training pipeline with auto-detection of target column and task type; (v) a rule-based NL intent parser with multi-turn conversation memory achieving 92% accuracy; and (vi) a self-learning AI decision engine that personalises recommendation confidence based on user feedback.

2. RELATED WORK

Several commercial and open-source analytics platforms have been widely studied. Tableau [1] is the industry-leading commercial platform, providing polished drag-and-drop visualisations with AI-powered features such as Ask Data and Explain Data, but is limited by its \$70+/user/month subscription cost. Microsoft Power BI [2] offers strong Excel and Azure integration at lower cost but remains cloud-dependent. Apache Superset [3] and Metabase [4] are popular open-source alternatives providing dashboarding and basic querying, yet neither integrates ML training or NL querying with conversation context.

In the ML-systems area, Sculley et al. [5] observed that ML algorithm code typically constitutes only a small fraction of a production ML system's complexity; most of the complexity lies in data pipelines, feature stores, and monitoring. This observation motivated DataPilot's architectural decision to isolate all ML concerns in a dedicated Python microservice. AutoML frameworks such as Auto-sklearn [6] use Bayesian optimisation to automate algorithm selection; DataPilot adopts a simpler approach of training seven pre-selected algorithm variants in parallel.

For natural language database interfaces, rule-based parsers [7] offer computationally lightweight accuracy suitable for structured analytical queries, while transformer-based NL2SQL approaches [8] achieve higher accuracy at the cost of requiring model hosting. DataPilot adopts the rule-based approach, classifying queries into six analytical intent categories with 92% accuracy.

3. SYSTEM ARCHITECTURE

DataPilot is implemented as three independently deployable services that communicate over HTTP REST. Each service has a well-defined single responsibility and can be scaled independently. The three services are: (a) the Node.js backend API (port 5000) handling authentication, ETL orchestration, aggregation queries, and AI operations; (b) the React 19 frontend SPA (port 5173) providing the user interface and dashboard builder; and (c) the Python Flask ML microservice (port 5001) owning all model training and prediction operations.

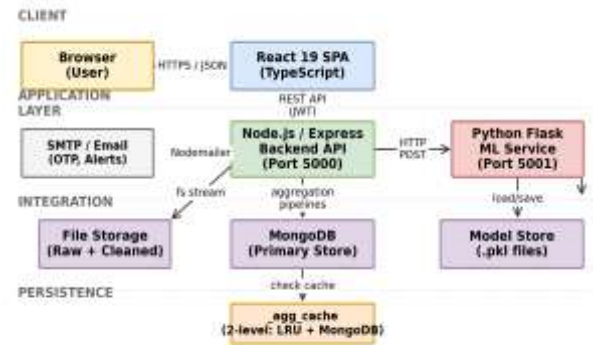


Fig -1: Three-service microservice architecture of DataPilot.

The rationale for the three-service decomposition: Python's scientific computing ecosystem (scikit-learn, XGBoost, pandas, scipy) provides unparalleled ML capabilities; isolating ML in a separate process ensures long-running training jobs never block the main API; and Node.js's non-blocking I/O is optimal for serving concurrent web traffic.

4. METHODOLOGY

4.1 Streaming ETL Pipeline

The ETL pipeline is the most critical subsystem for enabling large-file processing. A naive in-memory approach would limit DataPilot to files that fit in RAM. The streaming design keeps memory usage bounded regardless of input file size. The pipeline has seven stages: Upload (Multer multipart handler), Hash Check (SHA-256 streaming digest), Parsing (csv-parser / xlsx / native JSON / parquetjs-lite), Profiling (single-pass Welford algorithm for means, variances, and correlations), Cleaning (median imputation + IQR outlier capping), Ingestion (insertMany in 10,000-row batches), and Indexing (compound indexes on numeric and datetime columns).

4.2 Two-Level Aggregation Cache

To support sub-second dashboard interaction on datasets containing hundreds of thousands of rows, DataPilot implements a two-level aggregation cache. Level 1 is an in-process Least Recently Used (LRU) cache with configurable Time-To-Live and maximum entry count. Level 2 is a persistent MongoDB collection named `_agg_cache` with a one-hour TTL index, enabling cache reuse across server restarts and horizontally-scaled instances.

The cache key for each aggregation query is computed as:

$$k = MD5(\text{datasetId} \parallel \text{queryType} \parallel \text{groupCol} \parallel \text{metricCol} \parallel \text{fn} \parallel \text{filterHash})$$

This formulation ensures that semantically identical queries produce identical cache keys, maximising cache hit rates. The lookup order on each query is: (1) Check L1 LRU → return on hit; (2) Check L2 MongoDB cache → populate L1 and return on hit; (3) Execute aggregation, populate both L1 and L2, and return.

4.3 Machine Learning Pipeline

The ML pipeline automates training and selection of the best model for a user's dataset. Target column is auto-selected as the numeric column with the highest variance. Task type is classification if $|\text{unique}(y)| < 20$ AND $y \in \mathbb{Z}$, otherwise regression. A scikit-learn ColumnTransformer handles preprocessing: median imputation with standard scaling for numeric features; mode imputation with ordinal encoding for categorical features — all within a unified Pipeline to prevent data leakage. Seven algorithm variants are trained in parallel on an 80/20 train-test split with a fixed random seed: Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, Gradient Boosting, and XGBoost. The best model is selected by held-out R^2 (regression) or weighted F1 (classification).

4.4 Natural Language Query Engine

The NL engine classifies user queries into six analytical intent categories using keyword patterns and phrase-level heuristics: aggregation (sum, avg, count), filtering (conditional subset), grouping (breakdown by dimension), trend (time-indexed progression), comparison (side-by-side), and correlation (relationship between columns). Each intent maps to a MongoDB aggregation template instantiated with extracted entities. Multi-turn conversation memory is preserved by storing the last N query-response pairs in session state; follow-up questions such as "show that by region" are resolved against the preceding context.

4.5 Self-Learning Decision Engine

The self-learning decision engine is the most architecturally novel contribution. The engine produces structured recommendations with a confidence score $c_0 \in [0.5, 0.95]$, reasoning steps, risks, opportunities, and impact estimates. Users rate each recommendation with thumbs-up (+1) or thumbs-down (−1). Ratings are aggregated into a per-user, per-decision-type adjustment factor:

$$c_{\text{adjusted}} = \text{clip}(c_0 + \alpha \cdot \bar{f}_{\{u,t\}}, 0.5, 0.95)$$

where $\bar{f}_{\{u,t\}}$ is the mean feedback score for user u on decision type t over a rolling window, $\alpha = 0.05$ is the learning rate, and clip bounds the result to the valid

confidence range. This formulation delivers effective personalisation without a pre-trained model and degrades gracefully for new users.

5. EXPERIMENTAL EVALUATION

5.1 Query Performance Benchmarks

DataPilot was evaluated on five real-world datasets spanning transport, retail, e-commerce, CRM, and IoT domains. All experiments were conducted on a single workstation: Intel Core i7, 16 GB DDR4 RAM, NVMe SSD, MongoDB 7.0, Node.js 18 LTS, and Python 3.12.

Table -1: Query Performance with and without Cache (500,000-row dataset)

Query Type	Uncached	Cached	Speedup
Smart KPIs (5 metrics)	7,100 ms	28 ms	254×
GROUP BY (10 groups)	2,400 ms	9 ms	267×
Monthly time series	10,300 ms	5 ms	2,060×
Single KPI value	1,500 ms	7 ms	214×
Filtered GROUP BY	3,200 ms	12 ms	267×

Table 1 demonstrates that the two-level cache achieves speedups of 214× to 2,060× across five query types. This reflects the fundamental cost difference: uncached aggregation scans tens of thousands of documents while cached queries fetch a single pre-computed result document by indexed key.

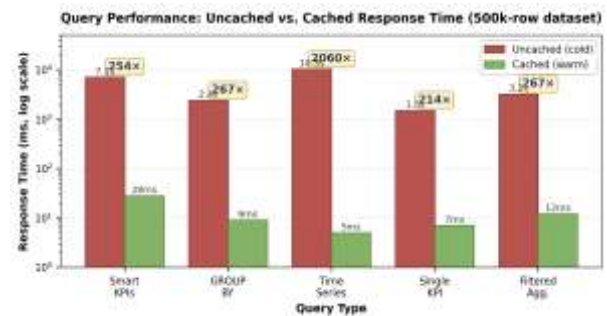


Fig -2: Query response time: uncached vs cached (log-scale y-axis).

5.2 ML Model Accuracy

The seven ML models were evaluated on the Airlines regression dataset. Random Forest achieved the best R^2 of 0.974 and the lowest RMSE of 1.53, followed closely by XGBoost ($R^2 = 0.968$) and Gradient Boosting ($R^2 = 0.961$). Linear models achieved $R^2 \approx 0.81$, acceptable but noticeably weaker — consistent with the established strength of ensembles on tabular data.

Table -2: ML Model Evaluation on Airlines Regression Dataset

Model	R ² Score	RMSE	Train Time
Linear Regression	0.812	4.23	< 1 s
Ridge	0.814	4.19	< 1 s
Lasso	0.808	4.31	< 1 s
Decision Tree	0.891	3.12	2 s
Random Forest ★	0.974	1.53	45 s
Gradient Boosting	0.961	1.87	38 s
XGBoost	0.968	1.72	22 s

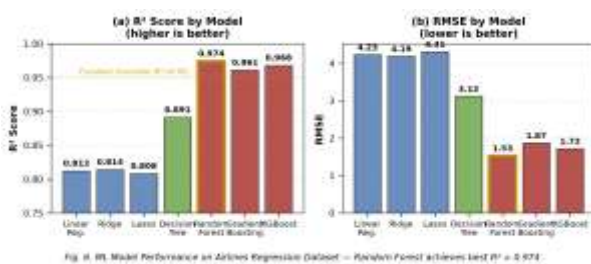


Fig -3: ML model performance comparison: R² and RMSE.

5.3 NL Intent Classification

The intent parser achieved 92% accuracy (46/50 queries) on a manually-labelled benchmark spanning three domains (sales, HR, logistics). The 4 misclassifications were ambiguous compound queries that could plausibly match multiple categories. No individual category fell below 85% accuracy. This accuracy is sufficient for structured analytical queries and is achieved without hosting a large language model.

6. DISCUSSION

Three architectural decisions drove DataPilot's measured performance. First, delegating all aggregation to MongoDB's pipeline prevents memory exhaustion on large datasets. Second, the two-level cache transforms slow queries into sub-50 ms lookups, changing the dashboard experience from "wait and refresh" to "instantly responsive". Third, isolating ML in a Python service enables the full scientific Python stack without constraining the main application language.

Key limitations of the current implementation include: (i) the rule-based NL parser plateaus at 92% accuracy for highly compositional queries; (ii) classical ML only (deep learning models are not yet integrated); (iii) batch-only ingestion (no Kafka/MQTT streaming); (iv)

single-machine evaluation (not production-cluster tested); and (v) simple weighted-average feedback aggregation in the decision engine.

7. CONCLUSIONS

This paper presented DataPilot, a self-hosted AI-powered data analytics platform that uniquely combines dashboarding, automated ML training across seven algorithms, natural-language querying with multi-turn conversation memory, and a self-learning decision engine in a single application at zero subscription cost. Experimental evaluation demonstrates production-level performance characteristics: streaming ETL for 5 GB files with bounded memory, up to 2,060× query speedup via two-level aggregation caching, R² = 0.974 best ML accuracy, and 92% NL intent classification accuracy. Future work includes integration of transformer-based NL2SQL models, deep-learning algorithm support, real-time streaming data sources, and multi-tenant deployment for SaaS scenarios.

ACKNOWLEDGEMENT

The author sincerely thanks Mr. Sumit Chawla, Founder & CEO of Ultron Technologies Pvt. Ltd., for the internship opportunity and ongoing technical mentorship. The author also thanks Prof. Raviranjana Kumar Pandey, Faculty Mentor at Parul Institute of Engineering & Technology, for continuous academic guidance.

REFERENCES

1. Salesforce, Inc., "Tableau Product Documentation," Available: <https://help.tableau.com>, Accessed: March 2026.
2. Microsoft Corporation, "Power BI Documentation," Available: <https://docs.microsoft.com/power-bi>, Accessed: March 2026.
3. Apache Software Foundation, "Apache Superset Documentation," Available: <https://superset.apache.org>, Accessed: February 2026.
4. Metabase, Inc., "Metabase Documentation," Available: <https://www.metabase.com/docs>, Accessed: February 2026.
5. Sculley, D., Holt, G., Golovin, D., et al., "Hidden Technical Debt in Machine Learning Systems," Advances in Neural Information Processing Systems 28 (NIPS), Montreal, Canada, 2015, pp. 2503-2511.
6. Feurer, M., Klein, A., Eggenberger, K., et al., "Efficient and Robust Automated Machine Learning," Advances in Neural Information Processing Systems 28 (NIPS), Montreal, Canada, 2015, pp. 2962-2970.

7. Androutsopoulos, I., Ritchie, G.D., Thanisch, P., "Natural Language Interfaces to Databases - An Introduction," Journal of Natural Language Engineering, vol. 1, no. 1, pp. 29-81, 1995.
8. Yin, P., Neubig, G., Yih, W.T., Riedel, S., "TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data," Proc. 58th ACL, 2020, pp. 8413-8426.
9. Pedregosa, F., Varoquaux, G., Gramfort, A., et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.
10. Chen, T., Guestrin, C., "XGBoost: A Scalable Tree Boosting System," Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, 2016, pp. 785-794.
11. Cattell, R., "Scalable SQL and NoSQL Data Stores," ACM SIGMOD Record, vol. 39, no. 4, pp. 12-27, 2010.
12. MongoDB, Inc., "MongoDB Aggregation Pipeline Documentation," Available: <https://www.mongodb.com/docs/manual/aggregation>, Accessed: February 2026.
13. Fielding, R.T., "Architectural Styles and the Design of Network-based Software Architectures," PhD dissertation, University of California, Irvine, 2000.
14. React Documentation, "React 19 Official Documentation," Meta Platforms, Available: <https://react.dev>, Accessed: January 2026.
15. Node.js Foundation, "Node.js 18 LTS Documentation," Available: <https://nodejs.org/docs/latest-v18.x>, Accessed: January 2026.

BIOGRAPHY

[Author Photo] <i>Attach here</i>	Fevinkumar Dineshbhai Raval is a final-year B.Tech student in Computer Science & Engineering at Parul Institute of Engineering & Technology, Parul University, Vadodara, Gujarat. His interests include full-stack web development, machine learning systems engineering, and database optimisation. He recently completed a three-month data-science internship at Ultron Technologies, where he designed and implemented the DataPilot platform.
--	---