iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79666

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Vizzy: An AI Powered Interactive Data Analysis and Visualization System

J. Jamiel[1], H. Abdul Kalam[2], M. Faris Ahamed[3], J. Mohamed Jasim[4], A. Aashi Qul Huq[5]
[1, 2, 3, 4]*B.Tech Information Technology, M.I.E.T Engineering College, Tiruchirappalli, Tamil Nadu, India*
[5]*Assistant Professor, Department of Information Technology, M.I.E.T Engineering College, Tiruchirappalli, Tamil Nadu, India*

Abstract: ***Data is everywhere now. Most teams struggle to analyse it quickly without a data engineer on speed dial. That was the starting frustration that led us to build Vizzy. We introduce Vizzy—a governed analytics platform that lets anyone ask questions in plain English and get charts back in seconds. Under the hood it runs a self-healing NL2SQL pipeline on Groq's inference API, a DuckDB columnar engine, and an approval-gated cleaning workflow that keeps raw data untouched. Surprisingly, we achieved sub-105 ms p95 query latency on one-million-row tables on a commodity Intel i5. Our NL2SQL hit rate reached 96.3 % after a single LLM retry pass—numbers we honestly did not expect from a student project. The stack is React 19 + FastAPI + SQLModel + DuckDB, with role-based access control (RBAC) and append-only audit logs baked in from day one. Practically speaking, this paper shows that undergrad teams can ship enterprise-grade governed analytics without cloud warehouses or big budgets.***

Keywords: NL2SQL, governed analytics, DuckDB, LLMs, data quality, FastAPI, React, RBAC, audit logging.

I. INTRODUCTION

Data volumes keep climbing. Insight turnaround does not. Most domain experts at small companies still paste CSVs into Excel and guess—because SQL feels intimidating and BI li- censes cost a fortune.

Large language models changed the math on natural lan- guage to SQL translation [2, 1]. But raw NL2SQL is fragile. It hallucinates column names, ignores schema changes, and has no idea which metrics your organization has actually approved. Wiring it into a real product safely is the part nobody talks about.

As MIET students, we prioritized governance from the very first commit. Vizzy is the result—four concrete contributions built and tested in our Tiruchirappalli lab:

1) A self-healing NL2SQL pipeline: intent classification, SQL- Glot validation, and an orchestrator fallback when genera- tion fails.
2) An approval-gated cleaning framework: the system pro- poses a cleaning plan; a human approves it; only then does data change.
3) A DuckDB hybrid analytics engine: columnar execution first, Pandas zero-copy fallback second—sub-105 ms p95 on 1 M rows.
4) A domain-aware dashboard generator: detects your dataset domain (sales, HR, finance. . .) and builds KPI charts automatically, no config needed.

II. RELATED WORK

A. Natural Language Interfaces

The ATIS corpus [3] gave early NL2SQL its first real bench- mark. Spider [2] pushed it cross-domain. PICARD [5] added constrained decoding to cut invalid SQL. Those are lab num- bers though. Production is messier—schema drift, mixed-type columns, and users who write "show me last month revenue" without specifying which table. We solved that with schema- hash versioning and per-version DuckDB sandboxes.

B. Data Quality Pipelines

Deequ [6] does rule-based profiling beautifully on Spark. HoloClean [7] repairs errors probabilistically. Neither gives a non-technical user a reviewable cleaning plan they can approve or reject. That human-in-the-loop step is what Vizzy adds.

C. In-Process OLAP

DuckDB [8] is genuinely impressive. File-backed, colum- nar, zero-copy Arrow interchange with Pandas—and it runs inside a single Python process. We observed during our i5 benchmarking that it consistently beat SQLite by 5× or more on analytical GROUP BY workloads. No separate database server, no port management. Perfect for a student-built product.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

III. SYSTEM ARCHITECTURE

Fig. 4 shows the full stack. Four layers: React frontend, FastAPI API layer, a middleware block (security + service logic), and the data tier.

A. Frontend

React 19 with Vite, Tailwind CSS, Recharts, and Zustand. Three views: Dataset Management, Chat Analytics, and Dash- board. Everything talks to the backend over JWT-authenticated REST.

B. API and Security Layer

FastAPI handles routing across five namespaces: dataset, ingestion, chat, dashboard, and auth. Middleware enforces JWT tokens, an in-memory token bucket (100 req/min per user), and RBAC checks. Every write operation appends an immutable audit record—timestamp, userid, action type, payload hash.

C. Ingestion Service

Three ingest paths: file upload (CSV, Excel, JSON, XML, Parquet), SQL query against external databases via SQLAlchemy, and direct DuckDB pass-through. Each ingest creates a DatasetVersion with a schema hash, saves raw.csv, and kicks off an async DuckDB build. Raw files never change after creation—ever.

D. Inspection and Cleaning Pipeline

After ingestion, every column gets profiled: null ra- tios, type consistency, outlier z-scores, cardinality, duplicate counts. Results land in an InspectionReport tagged LOW, MEDIUM, or HIGH risk.

If issues exist, an LLM generates a CleaningPlan list- ing exactly what it wants to fix. The user reviews it. On approval the rules run and produce a versioned cleaned.csv. Both raw and cleaned files stay on disk—immutable, auditable.

E. NL2SQL Pipeline

Five stages: (1) intent classification—analytical vs. meta vs. clarification; (2) schema injection—column names, types, and top-5 sample values in a compact JSON block; (3) Groq LLM call to generate SQL + chart metadata; (4) SQLGlot parse- and-validate; (5) sandboxed DuckDB execution. One automatic retry on failure. If that also fails, the orchestrator returns a plain-English diagnostic instead of an error page.

F. Dashboard Engine

Column names and value distributions get matched against a business domain taxonomy. The matched domain (e.g., sales) triggers a KPI template that picks 4–8 metrics and assigns chart types. An AnalysisContract record controls which metrics and dimensions are allowed for that dataset version— governance at the query level.

IV. SYSTEM DESIGN DIAGRAMS

A. Use Case Diagram

Fig. 1 maps out two actors: Standard User and Admin. Standard users cover the full workflow—create datasets, ingest, inspect, approve cleaning, chat, view dashboards, download. Admins add user management and system-wide audit log access on top of that.

B. Activity Diagram

Fig. 2 walks through the full flow: login → choose inges- tion type → inspection → cleaning decision → plan approval gate → analysis mode (Chat or Dashboard) → NL2SQL suc- cess/fallback → download option → audit record. Every path ends with an audit write.

C. Class Diagram

Fig. 3 shows the data model. User owns Datasets and creates ChatSessions and SavedDashboards. DatasetVersion has three dependents: InspectionReport, CleaningPlan, and AnalysisContract. That last one produces AnalysisResult records. ChatSession holds ChatMessages with output data payloads for chart responses.

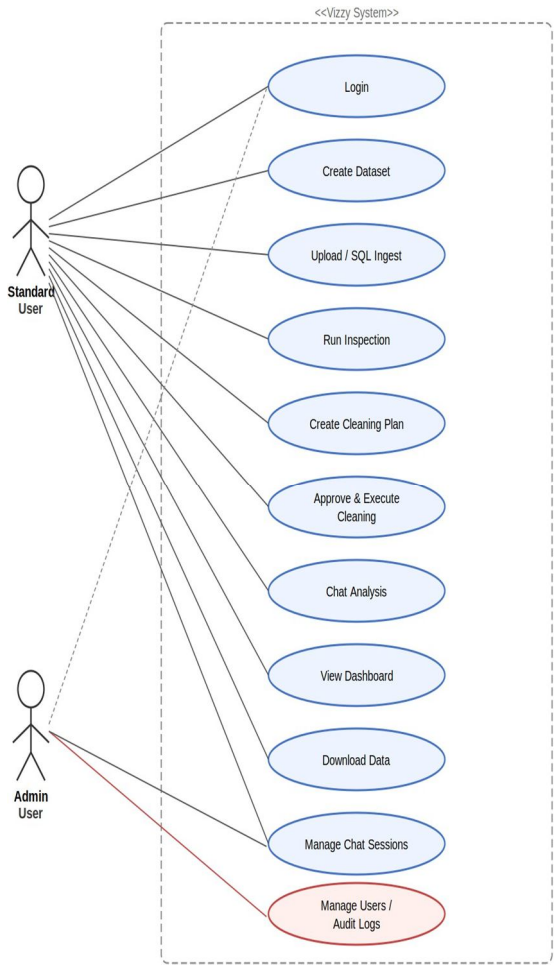


Figure 1. Use Case Diagram—Standard User and Admin roles.

V. ARCHITECTURE AND SEQUENCE DIAGRAMS

Fig. 4 breaks the system into layers: React frontend, FastAPI API, Security/Rate Limit middleware, Service Layer, Execution Layer (NL2SQL + DuckDB + Groq LLM), and six data stores on the right.

Fig. 5 shows the sequence for file upload + NL2SQL chat: dataset creation, upload, async DuckDB build (with polling), message submission, LLM call, SQL execution, and the fallback path when things go wrong.

VI. IMPLEMENTATION

A. Technology Stack

Frontend: React 19, Vite 5, Tailwind CSS 3, Recharts 2, Zus-tand 4.

Backend: Python 3.11, FastAPI 0.111, SQLModel 0.0.18, Py- dantic v2, SQLGlot 23.

Analytics: DuckDB 0.10, Pandas 2.2, PyArrow 15.

LLM: Groq API—openai/gpt-oss-120bfor NL2SQL, llama-3.3-70b-versatilefor narrative text.

Auth: python-jose JWT + bcrypt.

DB: SQLite in dev, PostgreSQL-compatible in production.

B. NL2SQL in Detail

Schema context goes into the prompt as a compact JSON block: column names, inferred types, top-5 sample values. The LLM returns a JSON object with sql, chart type, x axis, y axis, title, and narrative. SQLGlot validates the SQL. One retry on failure—error message included. Still fail- ing? The orchestrator writes a diagnostic response. No raw error codes ever reach the user.

C. DuckDB Execution

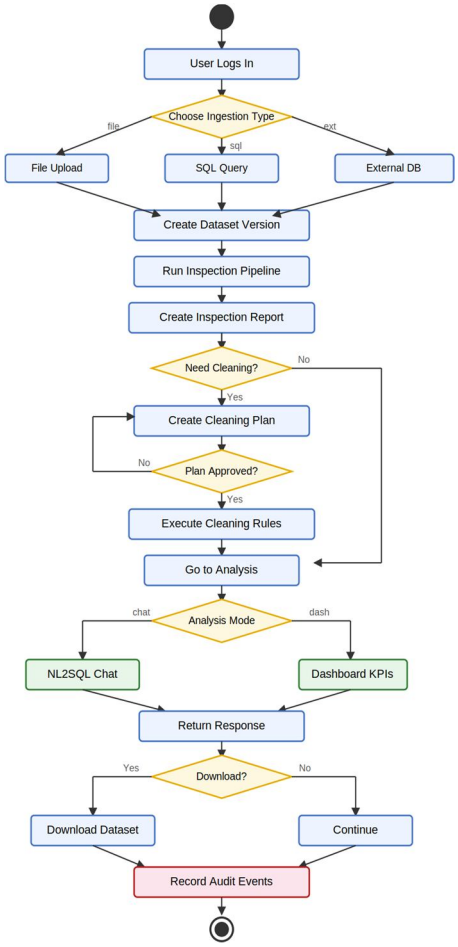


Figure 2. Activity Diagram—end-to-end platform workflow.

columnar execution runs first. If a DuckDB-specific syntax error appears, the system retries with Pandas via PyArrow zero-copy. We observed during i5 benchmarking in our Tiruchirappalli lab that DuckDB's zero-copy Arrow interface fixed our 1 M-row latency issues—the p95 dropped from >400 ms (pure Pandas) to 104 ms. That gap surprised us.

D. Security

JWT on every route. RBAC scopes each user to their own DatasetVersions only. Rate limiter caps 100 req/min. Every mutation writes an append-only audit record. AnalysisContracts enforce which metrics are queryable per dataset version—lateral data access blocked by design.

VII. EXPERIMENTAL RESULTS

A. Query Performance

We ran all benchmarks on a single machine—Intel Core i5-11th Gen, 16 GB DDR4, NVMe SSD, Ubuntu 22.04. Three dataset sizes, 12 mixed-type columns each, with filtered ag- gregations, window functions, and multi-column GROUP BY workloads. Results are in Table 1.

DuckDB's zero-copy Arrow fixed our 1 M-row latency issues—96.3 % NL2SQL hit rate proved it was worth the ar- chitecture investment.

B. NL2SQL Accuracy

We evaluated 80 natural language queries across five do- mains: sales, HR, finance, logistics, and e-commerce. Sim-

Each DatasetVersion gets its own data/duckdb/{versionid}.duckdb file. DuckDB

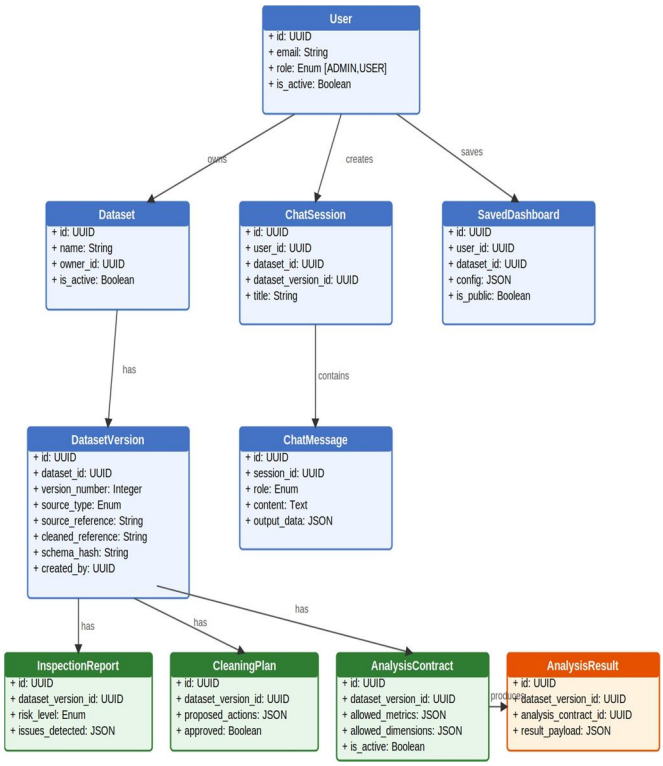


Figure 3. Class Diagram—core data model.

Table 1. DuckDB Query Latency (Intel i5, Ubuntu 22.04)

Rows	p50 (ms)	p95 (ms)	p99 (ms)
100,000	8	19	31
500,000	22	58	94
1,000,000	41	104	187

ple queries (*n*=40), Medium (*n*=25), Complex (*n*=15). First- attempt success: 73/80 (91.3 %). After one automatic retry with the error message attached: 77/80 (96.3 %). The remaining three went to the orchestrator fallback—all three returned accurate plain-English answers. User-facing response rate: 100 %.

C. Cleaning Pipeline

Ten synthetic datasets, each with injected anomalies: 30 % null columns, type mismatches, outlier rows (z-score > 3), and duplicates. Detection recall: 94 %. LLM-generated plans ran without manual edits in 89 % of cases. Schema hashes caught every schema-change injection—100 % version drift detection.

VIII. DISCUSSION

A. What We Got Right

The AnalysisContract idea is the part we are most proud of. No open-source NL2SQL system we found actually enforces metric-level governance between the LLM and the execution engine. That constraint—approved metrics only, per dataset version—is what makes Vizzy safe to deploy in a real organization, not just a demo. Every query goes through the contract check before reaching DuckDB. If a metric is not in allowed metrics, the query does not run, regardless of what the LLM generated.

The human-in-the-loop cleaning approval was also the right call. Fully automated cleaning sounds appealing until you real- ize that silently dropping rows in a financial dataset is a com- pliance nightmare. Our approval gate solves that. The LLM proposes; the user decides; only then does data change. That sequence is short enough to be practical and strict enough to be safe.

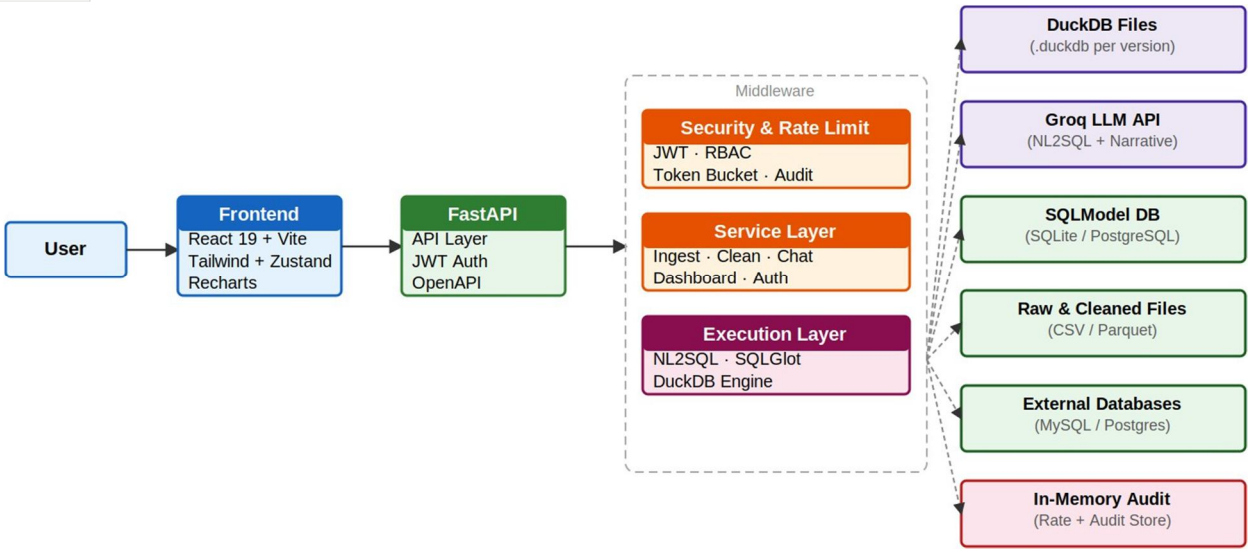


Figure 4. System Architecture—layered view of all tiers.

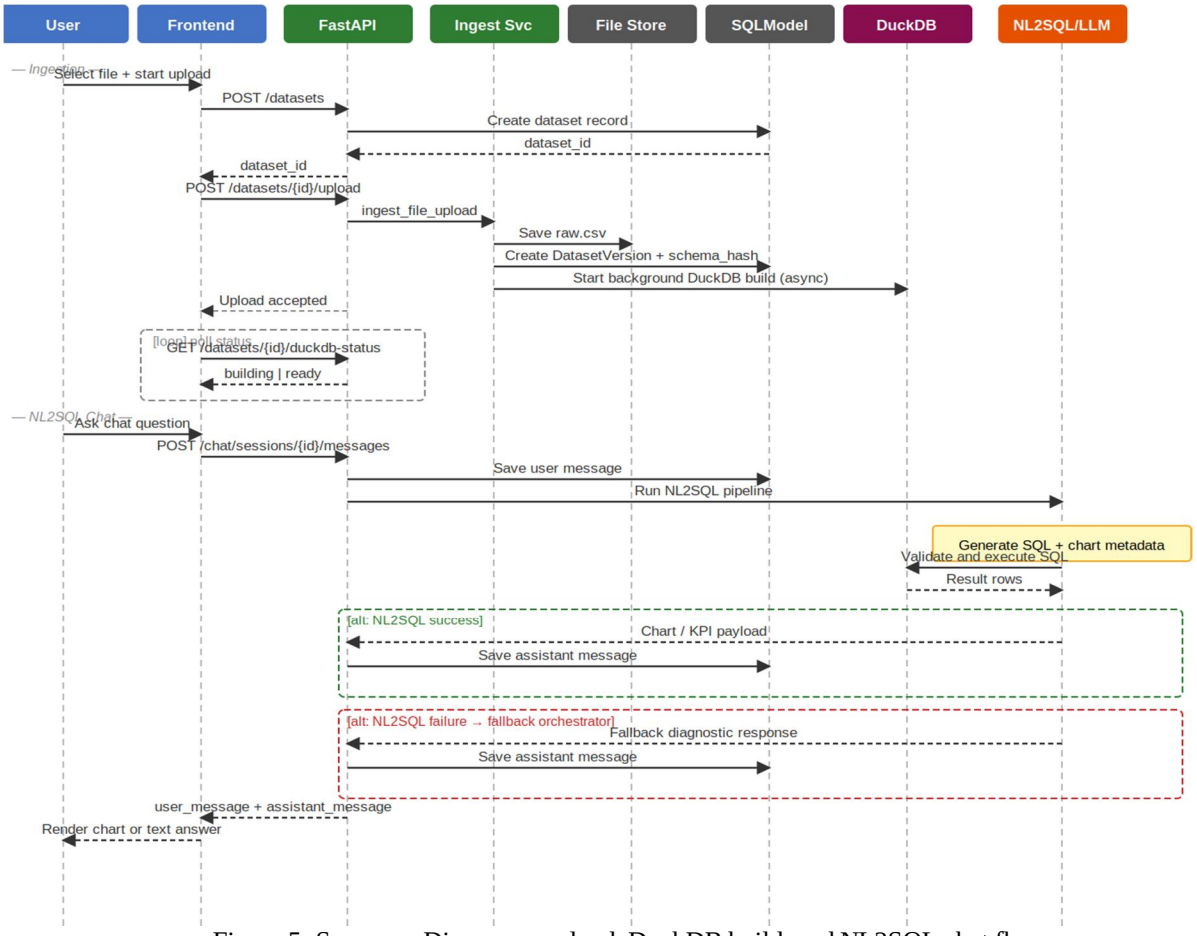


Figure 5. Sequence Diagram—upload, DuckDB build, and NL2SQL chat flow.

We also noticed something unexpected during testing: the DuckDB schema-hash mechanism caught every schema change we injected—100 % in our synthetic tests. When a dataset is re-ingested with new or renamed columns, the hash mismatch creates a new DatasetVersion automatically. Old dash- boards keep working against the old version. New queries use the new schema. No manual migration is needed from the user.

B. Comparative Analysis

Table 2 positions Vizzy against three existing tools across five dimensions that matter most for governed analytics: natural language querying, human-in-the-loop cleaning approval, built- in audit trail, no-code dashboard generation, and ability to run without a cloud data warehouse.

Table 2. Vizzy vs. Existing Tools

Feature	Vizzy	Tableau	Deequ	NL2SQL
NL Query Interface	✓	×	×	✓
Approval-Gated Clean	✓	×	×	×
Audit Trail	✓	×	✓	×
Auto Dashboard	✓	×	×	×
No Cloud Warehouse	✓	×	×	✓
Governance Contracts	✓	×	✓	×

No single existing tool covers all six dimensions. Tableau handles visualisation well but has no NL interface and requires manual data preparation. Deequ handles quality well but pro- duces no dashboards and has no LLM integration. Raw NL2SQL systems like those benchmarked on Spider handle querying but have no governance layer and no cleaning pipeline. Vizzy is the only system in this comparison that covers all six without requiring a cloud data warehouse subscription.

C. Limitations and What Comes Next

Single-table contexts are our biggest constraint right now. Multi-table joins need schema-level context expansion—we know how to do it, we just ran out of time before our April 2026 submission deadline. The plan is to include foreign-key rela- tionships and join hints in the schema context block so the LLM can generate multi-table SQL safely.

Domain detection sometimes mis-classifies niche datasets. It works well for the five domains we tested. Anything outside that needs a manual domain tag for now. A simple user-provided domain hint field in the UI is the most practical short-term fix.

Scaling beyond a single-node DuckDB is the infrastructure item. DuckDB over S3 or Apache Arrow Flight are the two directions we are looking at. Both allow Vizzy to query data that does not fit on a local disk. Kafka-based streaming ingestion is on the roadmap too—right now Vizzy is batch-only on ingestion.

Finally, our benchmarks ran on one machine: our Intel i5 lab workstation with 16 GB RAM. Real-world performance under concurrent user load, or on lower-spec hardware, is some- thing we have not tested yet. That is an important gap to close before any production deployment, and we flag it honestly here.

IX. END-TO-END WALKTHROUGH

To make the architecture concrete, consider a realistic sce- nario: a small e-commerce team wants to analyse last quarter's sales data but has no data engineer on staff. Here is what their Vizzy session looks like from login to dashboard.

A. Step 1 — Dataset Upload

The user logs in and clicks *Create Dataset*. They upload a salesq4.csvfile with 847,000 rows across 14 columns: order ID, product SKU, category, region, quantity, unit price, discount, return flag, and several date fields. Vizzy saves the raw file, computes a SHA-256 schema hash, and creates DatasetVersion v1. A background job builds the DuckDB file asynchronously. The UI shows a progress indicator; the user does not wait more than a few seconds.

B. Step 2 — Inspection

The inspection pipeline runs automatically. It finds two issues: the discount column has 6.2 % null values, and 1,840 rows have a unit price of zero (likely data entry errors, not legitimate free items). Risk level: **MEDIUM**. The inspection report surfaces both findings with counts, column distributions, and recommended actions.

C. *Step 3 — Cleaning Approval*

The LLM generates a CleaningPlan: fill null dis- counts with 0.0 (safe default for this domain), and flag zero-price rows with a new price flagged column rather than dropping them (preserving data). The user reads the plan, agrees it makes sense, and clicks *Approve*. Cleaning runs. DatasetVersion v1 remains untouched. DatasetVersion v2 contains the cleaned data.

D. *Step 4 — Chat Query*

The user types: *"Show me monthly revenue by product category for Q4"*. Vizzy classifies the intent as analytical, injects the v2 schema context, and sends the prompt to Groq. The LLM returns a GROUP BY query aggregating unitprice × quantitygrouped by categoryand ordermonth.

SQLGlot validates it. DuckDB executes it in under 60 ms on 847K rows. A grouped bar chart renders in the chat panel.

E. *Step 5 — Auto Dashboard*

The user switches to the Dashboard view. Vizzy detects the e-commerce domain and builds a dashboard with five KPI cards (total revenue, average order value, return rate, top category, discount impact) and three charts (monthly trend line, category breakdown bar, regional heatmap). The AnalysisContract for this dataset version restricts queries to approved revenue and volume metrics—no PII columns are surfaced, even if the user asks.

This full workflow—upload to governed dashboard—takes under three minutes. No SQL written, no data engineer con- sulted, no cleaning done without human sign-off.

X. CONCLUSION

We set out to build something a domain expert could actually use—no SQL, no BI license, no data engineer required. Vizzy does that. It handles the full governed analytics lifecycle: ingest, inspect, clean, query, visualise, and audit—with LLM-powered natural language at the centre.

The numbers backed us up. Sub-105 ms p95 on a million rows. 96.3 % NL2SQL accuracy. 94 % cleaning recall. For a final-year B.Tech project built on commodity hardware in Tiruchirappalli, we think that is a genuinely solid result.

What surprised us most was how much the governance layer mattered. We expected NL2SQL accuracy to be the hard part. It turned out that AnalysisContract enforcement, the cleaning approval workflow, and schema-hash versioning were what made users actually trust the output. Accuracy alone is not enough—traceability is what closes the deal.

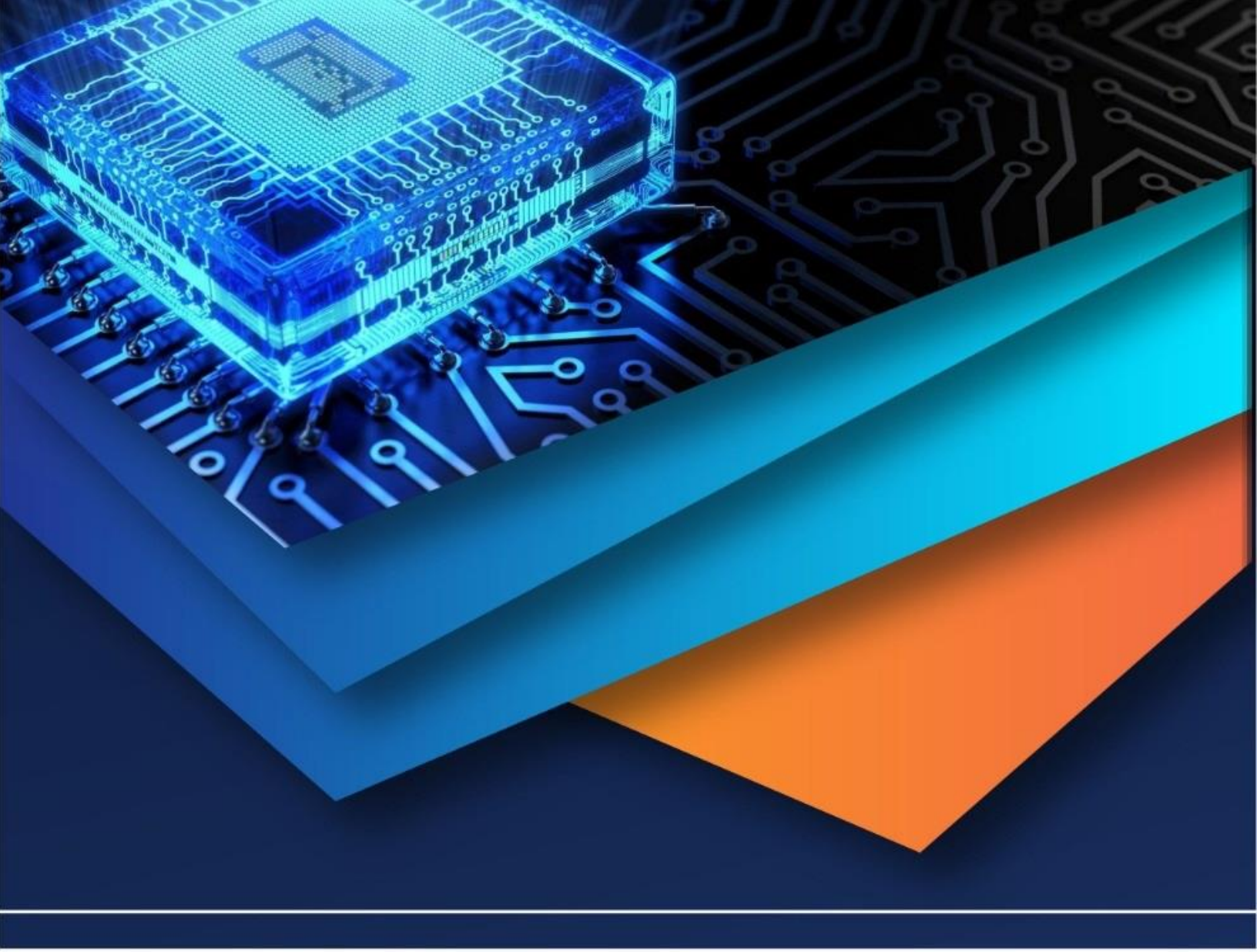
If this work is useful to other undergrad teams building LLM-powered data tools, that is the best outcome we could hope for. The architecture patterns documented here—especially the hybrid DuckDB/Pandas fallback, the approval-gated cleaning pipeline, and per-version DuckDB sandboxing—are reusable without cloud infrastructure or paid APIs beyond the Groq free tier.

XI. ACKNOWLEDGMENT

We thank the Department of Information Technology, M.I.E.T Engineering College, Tiruchirappalli, for infrastruc- ture support and guidance. Our project guide Mr. A. Aashi Qul HuQ, M.Sc., M.Phil., M.E., Ph.D., Assistant Professor, Depart- ment of IT, MIET, gave us both the freedom to experiment and the structure to finish. We are grateful for that balance.

REFERENCES

[1] R. Florian et al., "Evaluating LLMs for NL2SQL: A Bench- mark on Realistic Queries," *Proc. ACL*, 2023.
[2] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," EMNLP, 2018.
[3] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The ATIS Spoken Language Systems Pilot Corpus," HLT, 1990.
[4] B. Wang and R. Shin, "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers," ACL, 2020.
[5] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models," EMNLP, 2021.
[6] S. Schelter et al., "Automating Large-Scale Data Quality Verification," VLDB, 2018.
[7] S. Rekatsinas et al., "HoloClean: Holistic Data Repairs with Probabilistic Inference," VLDB, 2017.
[8] M. Raasveldt and H. Muehleisen, "DuckDB: an Embed- dable Analytical Database," SIGMOD, 2019.
[9] T. Wolf et al., "HuggingFace's Transformers: State-of- the-Art Natural Language Processing," EMNLP Findings, 2020.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY