

Askify: A Privacy-First Conversational Data Assistant

A CSV Conversational Data Assistant

Team: Ahmed Javed · Muhammad Taha Atif · Ali Murtaza

Date: November 7, 2025



The Data Analysis Bottleneck & Askify's Solution

Current Challenges

- Analysts burdened by repetitive SQL/Pandas queries.
- Non-technical stakeholders lack direct insight access.
- Cloud LLM APIs raise significant privacy and cost concerns.
- Small teams often lack the dedicated GPUs for fine-tuning.

Our Solution: Askify

- An offline, privacy-preserving conversational interface for CSV data.
- Translates natural language to SQL/Pandas, then delivers results, explanations, and visualizations.
- Ensures no data leaves the device, eliminating cloud dependencies.



Askify's End-to-End RAG Pipeline Architecture

Askify employs a sophisticated Retrieval-Augmented Generation (RAG) pipeline to deliver accurate and contextually relevant responses, all while maintaining data privacy.



Key Architectural Components:

Local LLM Utilizes quantized Qwen/LLaMA models for efficient on-device processing.	RAG System Leverages SBERT for semantic embeddings and FAISS for rapid vector search.	SQL Generation & Validation Intelligently generates and rigorously validates SQL queries for accuracy.
Sandboxed Execution Executes queries within a secure, isolated environment.	Natural Language Explanations Provides clear, concise explanations of results to enhance user understanding.	

Core Technologies: A Privacy-First, Local-First Stack

Askify is built upon a robust stack of technologies, carefully selected for their ability to ensure privacy, performance, and local-first operation.

Embedding	Sentence-BERT (all-MiniLM-L6-v2)	Generates semantic representations of CSV rows for intelligent context matching.
Vector DB	FAISS	Enables efficient similarity search for fast and relevant data retrieval.
LLM	Qwen2.5-0.5B-Instruct + LoRA	Powers SQL generation, enhanced with fine-tuning for specific data tasks.
Quantization	bitsandbytes (4-bit)	Significantly reduces model size, enabling efficient local inference on consumer hardware.
Fine-tuning	QLoRA on Kaggle GPUs	Adapts the base model to specific domains without requiring dedicated, expensive hardware.
UI Framework	Streamlit	Provides an interactive and user-friendly web interface for seamless interaction.
Execution	SQLite + Pandas	Ensures secure and sandboxed execution of generated SQL queries.

RAG Implementation: From Question to Insight

Askify's Retrieval-Augmented Generation (RAG) system efficiently transforms natural language questions into accurate data insights, all within a privacy-first, local environment.



Data Preparation

- CSV parsed into schema, stats, and sample chunks.
- Chunks converted to semantic embeddings via Sentence-BERT.
- Embeddings stored in a FAISS vector index for rapid search.



Query Understanding & Retrieval

- User question embedded to query FAISS index.
- Top relevant data chunks retrieved and deduplicated.
- Context built from retrieved chunks for the LLM.



SQL Generation & Validation

- LLM (Qwen + LoRA) generates SQL based on context and prompt.
- Generated SQL validated for syntax, column existence, and safety.

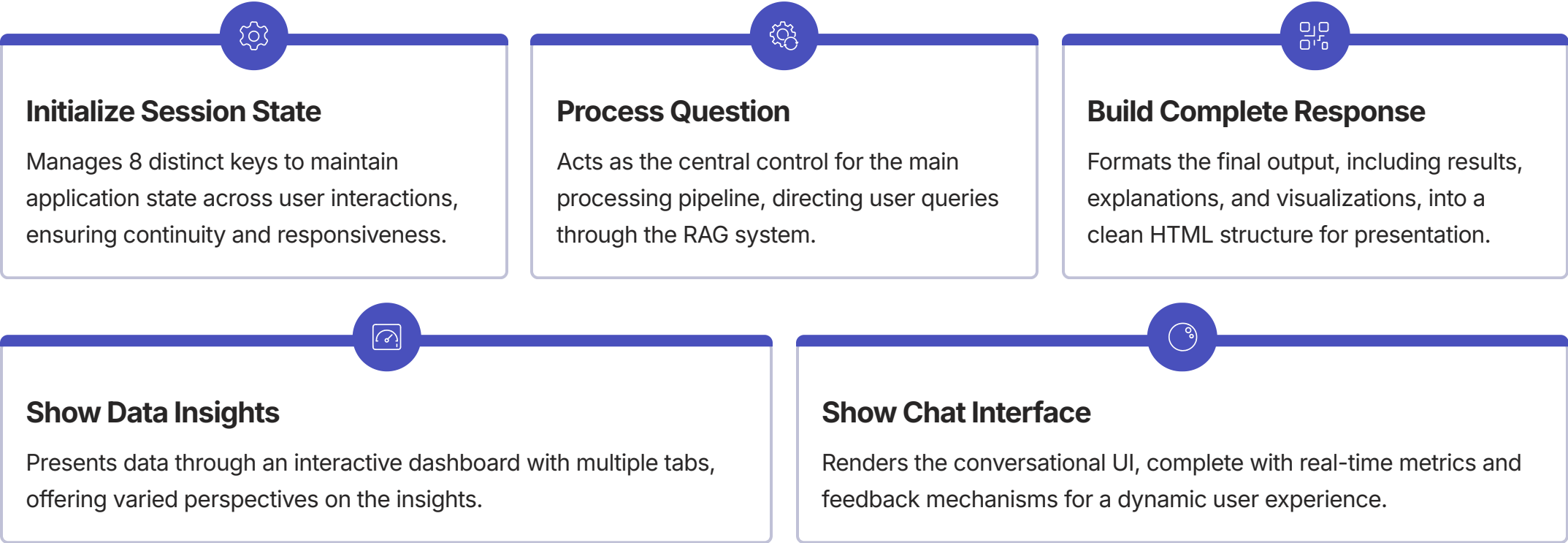


Execution & Explanation

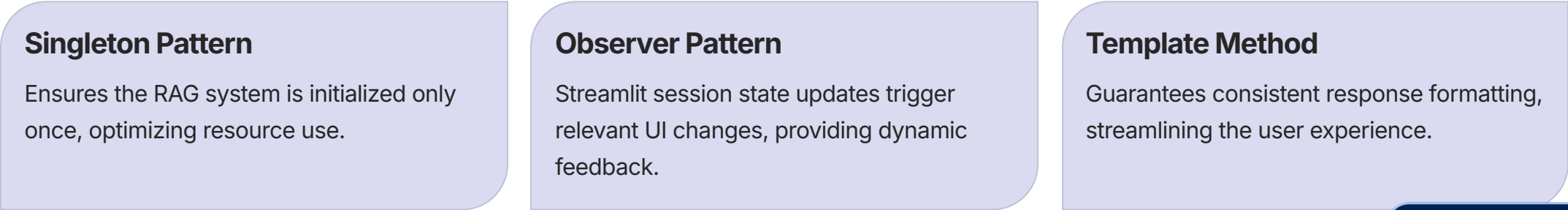
- Validated SQL executed in a sandboxed SQLite environment.
- Results formatted and explained in natural language.

Deep Dive: Frontend & Orchestration ([app.py](#))

The `app.py` serves as the main application hub, orchestrating user interactions and backend processes to deliver a fluid experience.



Key Design Patterns:



Deep Dive: AI Model & SQL Generation

The intelligence behind Askify's query generation resides in `qwen_client.py` and `prompt_engineer.py`, working in concert to translate natural language into precise SQL.

`qwen_client.py`: Model Wrapper

→ `load_model()`

Manages a 3-step loading process for the LLM, integrating LoRA adapters for fine-tuned performance.

→ `generate_response()`

Orchestrates the entire SQL generation pipeline, taking the user query and context to produce a candidate SQL statement.

→ `_extract_and_clean_sql()`

Removes extraneous markdown and prefixes from the LLM's output to isolate the pure SQL query.

→ `_fix_sql_structure()`

Applies heuristics to correct common structural issues in generated SQL, such as WHERE-FROM clause ordering.

`prompt_engineer.py`: Prompt Construction

→ `create_prompt()`

Builds a structured 5-section prompt, guiding the LLM with clear instructions, context, and examples.

→ `needs_joins_adapter()`

Employs keyword detection to identify when complex queries requiring SQL JOINS are needed, adapting the prompt accordingly.

→ `validate_and_clean_sql()`

Performs final syntax validation and cleaning of the generated SQL, ensuring it is executable and robust.

Deep Dive: Data Processing & Retrieval

Askify's retrieval mechanism, powered by `csv_chunker.py` and `vector_store.py`, ensures relevant data is efficiently found and delivered for contextual understanding.

`csv_chunker.py`: Intelligent Chunking

- **`chunk_dataframe()`**

Divides large CSV dataframes into 4 distinct chunk types, optimizing for varied information retrieval needs.

- **`_create_schema_chunk()`**

Generates chunks containing column names and their respective data types, crucial for accurate SQL generation.

- **`_create_statistical_chunks()`**

Extracts and chunks key statistical summaries (mean, std, min, max) for numerical columns, enriching context.

- **`_create_sample_chunks()`**

Creates chunks of sample values from columns, providing concrete examples for the LLM to understand data distribution.

- **`_create_column_chunks()`**

Processes per-column metadata, offering a detailed understanding of each feature within the dataset.

`vector_store.py`: FAISS Search

- **`add_chunks()`**

Normalizes and stores chunk embeddings into the FAISS index, preparing them for rapid retrieval.

- **`search()`**

Implements a 2-stage search: first converting text queries to embeddings, then using FAISS for similarity search.

- **Strategy Pattern**

Allows for multiple search strategies, ensuring flexibility and optimization based on query type and data structure.

Deep Dive: SQL Execution & Explanation

Once SQL is generated, Askify focuses on secure execution and clear explanation, handled by `sql_executor.py` and `lama_explainer.py`.

`sql_executor.py`: Safe Execution

1 `execute_sql()`

Executes SQL queries securely within an in-memory SQLite database, preventing external data access or modification.

2 `_normalize_sql_query()`

Maps column names in the generated SQL to the actual column names in the CSV, ensuring query validity.

3 `_find_best_column_match()`

Employs 4 distinct matching strategies to robustly identify the correct column even with minor discrepancies.

`lama_explainer.py`: Results Explanation

1 `explain_results()`

Utilizes TinyLlama to generate natural language explanations of the query results, making data insights accessible.

2 `_prepare_simple_context()`

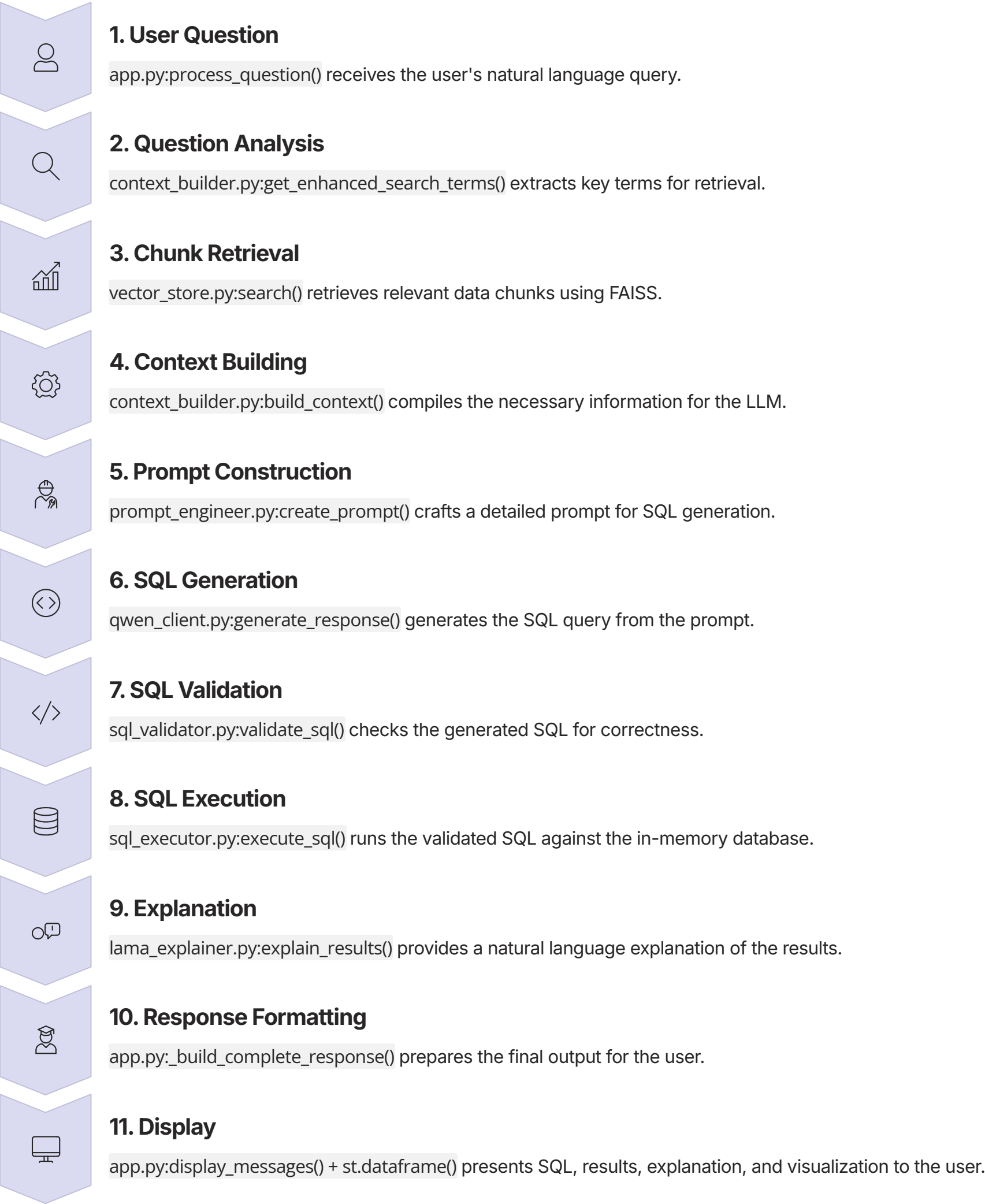
Prepares a concise and relevant context for the explanation model, ensuring accurate and focused output.

3 `Fallback Mechanism`

Includes a template-based fallback if the explanation model encounters issues, guaranteeing a user-friendly response.

Complete Data Flow: "Show top 5 customers by total sales"

This example illustrates the seamless journey of a user's query through the entire Askify pipeline, from natural language to actionable insights.



Output: SQL + Results + Explanation + Visualization

Results & Performance

Askify delivers significant quantitative and qualitative improvements in data interaction, outperforming baseline models and providing rapid, accurate insights.

Quantitative Improvements

Overall Score	0.207	0.185	+11.9%
Faithfulness	0.297	0.187	+58.8%
Query Wins	13	6	+117%
SQL Validity	31.8	27.3	+16.5%
Complex Query Success	71	29	+145%

Key Achievements

+365% Comparative Queries

Significant gains in handling questions involving comparisons between data points.

+265% Filtered Aggregations

Improved accuracy for queries requiring aggregation with specific filters.

+59% Schema Understanding

Enhanced ability to correctly interpret and utilize database schema information.

User Interface & Experience

Askify is engineered for an intuitive and empowering user experience, making data exploration accessible to everyone.

Askify - AI Data Assistant

http://localhost:8501/

Askify - AI Data Assistant

http://localhost:8501/

Upload Your Data

Choose a CSV file

Drag and drop file here

Limit 200MB per file • CSV

Browse files

Exam_Score_Pr...

1.4MB

How It Works

1. Upload your CSV file

2. Wait for AI processing

3. Ask questions naturally

4. Get instant SQL + insights

Example Questions

"Show top 10 customers"

"What's the average sales by category?"

"Find products with highest profit"

"Show monthly trends"

"Correlation between age and spending"

System Status

RAG System: Ready

AI Model: Loaded

SQL Validator: Ready

SQL Executor: Ready

Show Data Insights

Ask Anything About Your Data

I understand natural language - try questions like 'show top products by sales' or 'find customers with highest spending'

Total Rows

20000

Total Columns

13

Numeric Columns

6

Text Columns

7

You:

show data of students whose class attendance is greater than 90

Askify:

AI Insight

The query returns 3,307 rows (or records) of data the "Class Attendance" column has a value greater than or equal to 90%. This indicates that there were at least 3,307 students who attended their classes for at least 90% of the scheduled hours during the specified time period.

Query Executed Successfully

Found 3307 results

View SQL Query

SELECT student_id, class_attendance FROM data WHERE class_attendance > 90

</div>

Query Results:

student_id	class_attendance
81	467
82	479
83	490
84	491
85	499
86	504
87	512
88	513
89	527
90	529

2 of 39

See how I found this answer

Relevant Data Context:

Question: show data of students whose class attendance is greater than 90

Schema:

SCHEMA INFORMATION:

Dataset has 20000 rows and 13 columns

Columns and their data types:

student_id: int64 (0 missing values)

Top Retrieved Information:

Chunk 1 (Similarity: 0.391)

NUMERICAL COLUMNS STATISTICS:

student_id:

Mean: 10000.50

Std: 5773.65

Min: 1.00

Max: 20001.00

age:

Mean: 20.47

Std: 2.28

Min: 17.00

Max: 24.00

study_hours:

Mean: 4.01

Std: 2.31

...

Chunk 2 (Similarity: 0.391)

DATASET SCHEMA AND SAMPLE VALUES:

=====

Column: student_id

Type: int64

Range: 1.00 to 20001.00

Common values: 1, 2, 3

Column: age

Type: int64

S...

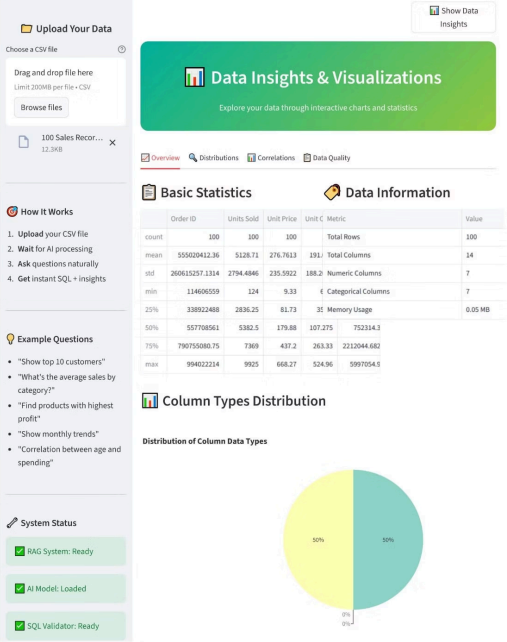
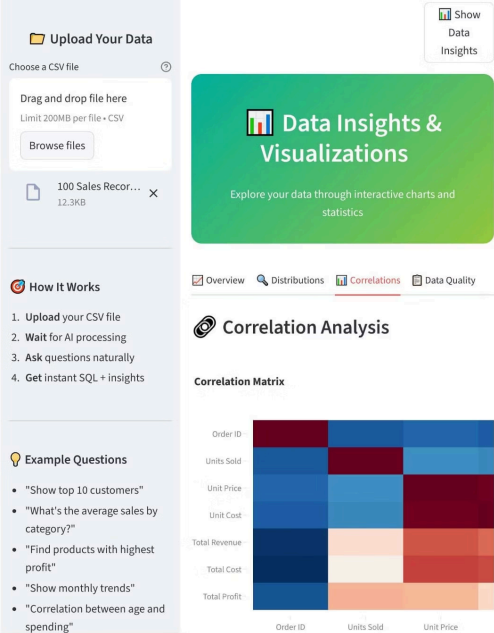
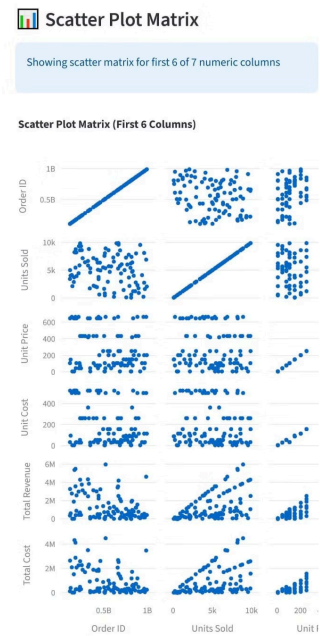
Chunk 3 (Similarity: 0.153)

14/12/2025, 5:36 am

1 of 39

14/12/2025, 5:36 am

Made with GAMMA



Kaggle Training Workflow

Leveraging Kaggle's free GPU resources, Askify employs a robust and cost-effective training pipeline for continuous model improvement.

1. Data Preparation

- Synthetic 16000 Q/A pairs
- Public datasets (customer, UCI, sales data)
- CSV-centered instruction datasets

2. QLoRA/LoRA Training

- Base model: Qwen2.5-0.5B-Instruct
- Adapter training on Kaggle notebooks
- Batch sizes tuned to available VRAM

3. Export & Integration

- Export LoRA adapter weights
- Convert to GGUF/8-bit formats
- Integrate into local inference stack

Key Benefits

Zero-Cost Fine-Tuning

Achieve high-quality model fine-tuning without incurring GPU infrastructure costs.

Sufficient VRAM





Kaggle's T4/P100 GPUs provide ample VRAM for 4-bit quantized training, enabling complex tasks.

Reproducible Workflows

Leverage notebook-based training for fully reproducible and shareable fine-tuning processes.

Askify: Areas for Improvement - The 4 Key Pillars

To evolve Askify from a robust prototype to an industry-leading data assistant, we've identified four key areas for strategic improvement.

<div></div> <div><h2>FUNCTIONALITY EXPANSION</h2><p>What Askify Can't Do Yet</p><h3>Multi-Data Capabilities</h3><ul style="list-style-type: none">Cross-file JOINS between multiple CSVsAdvanced analytics (statistical tests, time series, ML insights)Database connections beyond local CSV files<h3>Conversational Intelligence</h3><ul style="list-style-type: none">Clarification dialogues ("Did you mean monthly or quarterly?")Query refinement suggestionsContext memory across conversation sessions<h3>Output Enhancement</h3><ul style="list-style-type: none">Smarter visualizations with chart recommendationsMulti-format exports (PDF, Excel, embedded widgets)Collaborative sharing of insights and queries</div>	<div></div> <div><h2>PERFORMANCE & SCALABILITY</h2><p>Making Askify Faster & More Robust</p><h3>Speed Optimization</h3><ul style="list-style-type: none">Reduce 4-8 second latency through caching and lazy loadingHandle >5M rows with vector compression and shardingParallel processing for complex queries<h3>Reliability Improvements</h3><ul style="list-style-type: none">Increase SQL validity rate from 31.8% to 70%+Reduce 7.4% hallucination rate with better verificationBetter error recovery with intelligent fallbacks<h3>Scalability Features</h3><ul style="list-style-type: none">Multi-user support with role-based accessEnterprise deployment options (Docker, cloud, on-prem)API integration with existing BI tools</div>
<div></div> <div><h2>INTELLIGENCE UPGRADE</h2><p>Making Askify Smarter</p><h3>Model Enhancement</h3><ul style="list-style-type: none">Continuous learning from user correctionsDomain adaptation for specific industriesPersonal vocabulary learning for organizations<h3>Retrieval Improvement</h3><ul style="list-style-type: none">Hybrid search combining semantic + keyword + metadataBetter chunking strategies for complex schemasQuery expansion with synonyms and related terms<h3>Reasoning Capabilities</h3><ul style="list-style-type: none">Multi-step problem solving for complex questionsUncertainty quantification (confidence scores)Self-correction mechanisms</div>	<div></div> <div><h2>ENTERPRISE READINESS</h2><p>From Prototype to Production</p><h3>Security & Governance</h3><ul style="list-style-type: none">Data masking for sensitive informationAudit logging and compliance reportingQuery approval workflows for sensitive data<h3>DevOps & Maintenance</h3><ul style="list-style-type: none">Containerization for easy deploymentAuto-updates for models and adaptersHealth monitoring and alerting<h3>User Experience</h3><ul style="list-style-type: none">Onboarding tutorials and interactive guidesSkill-level adaptation (beginner vs. expert modes)Professional reporting with templates and branding</div>

Improvement Priorities at a Glance

This table outlines the phased approach to Askify's development, categorizing improvements by their impact and timeline.

	Quick Wins	High Impact	Strategic
NOW	Better error messages Query history Basic exports	-	-
NEXT	-	Multi-CSV JOINS Clarification dialogues Better charts	-
FUTURE	-	-	Enterprise deployment Continuous learning ML integration

Key Technical Patterns: Design Principles in Askify

Askify leverages established design patterns to ensure modularity, maintainability, and scalability across its complex architecture.

1

Pipeline Pattern

Implemented for sequential data processing: DataFrame → Chunker → Embedder → VectorStore, etc.

2

Strategy Pattern

Utilized for 4 distinct column matching strategies, enabling flexible column name resolution.

3

Template Method

Applied in SQL generation, using a base prompt with dynamic sections for consistent output.

4

Factory Pattern

Used for RAG system initialization and other component creations, promoting decoupled design.

5

Observer Pattern

Streamlit session state manages UI state by updating components dynamically upon changes.

6

Singleton Pattern

Ensures the RAG system instance is unique, optimizing resource management.