TabGuard: Agentic LLM Orchestration for Adaptive Tabular Anomaly Detection via Dynamic Validator Selection and Generation

Srihari Unnikrishnan and Minghua Ma
Microsoft Research
{sriunni,minghuama}@microsoft.com

Abstract

Tabular anomaly detection is challenging because real-world tables contain heterogeneous columns, ranging from structured identifiers to free-form text. Existing methods face a fundamental trilemma: rule-based systems require extensive manual configuration and fail on novel schemas; statistical methods scale efficiently but miss semantic errors; and LLM-based approaches understand semantics but incur prohibitive per-cell inference costs. No prior method simultaneously addresses semantic heterogeneity, domain-specific validation rules, and enterprise-scale processing.

We introduce TABGUARD, an agentic framework that resolves this trilemma through semantic routing. Using LLM function calling, the system analyzes a small sample of each column and dynamically selects the most effective validation strategy, routing to a regex-based validator for syntactic patterns, a code-generation validator for domain-specific rules (such as Luhn checksums for credit cards), or an embedding-based validator for distributional outliers. This architecture decouples expensive cognitive reasoning ($O(m)$ LLM calls for m columns) from scalable programmatic execution, enabling deployment on enterprise datasets without per-cell inference.

1 Introduction

Data quality is critical for organizations relying on tabular datasets for business intelligence and machine learning. However, real-world data frequently contains erroneous, inconsistent, or semantically invalid entries that can propagate through analytical pipelines (Choudhury et al., 2025). For instance, a single misplaced decimal point in a financial report can lead to significant monetary losses, while incorrect patient data in healthcare can have severe operational consequences. Column-level validation—checking that each value conforms to the expected format, domain rules, and distributional norms for its column—is a foundational component of data quality pipelines, and is the focus of this work.

Tabular data presents unique challenges for anomaly detection due to its inherent heterogeneity (Figure 1). We identify three primary challenges that motivate our design, each addressed by a specific component of our architecture:

Challenge 1: Semantic Heterogeneity. Real-world tables contain columns with fundamentally different semantics each requiring distinct validation logic. A regex that validates transaction IDs will fail on product descriptions; statistical outlier detection that works for numeric columns produces nonsensical results on categorical data. Existing monolithic approaches (Cao et al., 2023; Tsai et al., 2025) apply uniform detection strategies across all columns, fundamentally misaligning the detection method with data semantics. *Our solution:* An LLM agent that dynamically routes each column to the semantically appropriate validator.

Challenge 2: Domain Specificity. Many data types embed domain-specific validation rules that cannot be expressed through statistical properties or pattern matching alone. Credit card numbers require Luhn checksum verification; ISBNs need modulo-11 validation; temporal fields demand logical consistency checks. Prior constraint-learning approaches (Chen et al., 2025) are bounded by patterns observed in training corpora, while feature-based methods (Senaratne et al., 2023) cannot express procedural validation logic. *Our solution:* LLM-powered code generation that synthesizes arbitrary validation functions from world knowledge, enabling unlimited rule expressiveness.

Challenge 3: Scalability. Enterprise datasets routinely contain millions of rows across hundreds of columns. Per-cell LLM inference as employed by AnoLLM (Tsai et al., 2025) and similar approaches (Bendinelli et al., 2025) incurs prohibitive

Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026), pages 152–161

latency and cost, rendering these methods impractical for production deployment. *Our solution:* Amortized inference where the LLM reasons once per column ($O(m)$ calls) to generate deterministic validators that execute programmatically on all values without further LLM involvement.

Traditional anomaly detection approaches fall into two categories, each with significant limitations. Rule-based systems require extensive manual configuration for each data type and fail to adapt to novel schemas, making them impractical for organizations managing thousands of heterogeneous tables (Hulsebos et al., 2019). Statistical and machine learning methods, including isolation forests, one-class SVMs, and autoencoders (Cao et al., 2023), while more generalizable, treat all columns uniformly despite fundamental differences in what constitutes an anomaly across semantic types. These methods excel at detecting distributional outliers but lack semantic understanding, frequently flagging valid edge cases while missing semantically invalid entries that happen to be statistically consistent.

The emergence of Large Language Models (LLMs) presents an opportunity to bridge this gap through their broad world knowledge enabling semantic understanding of diverse data types (Biester et al., 2024; Bendinelli et al., 2025). However, directly applying LLMs to validate every cell in large datasets incurs prohibitive computational costs. Consider an enterprise table with 100,000 rows and 50 columns: validating each cell individually would require 5 million LLM API calls, translating to substantial latency and significant monetary cost. This scalability barrier renders per-cell LLM validation impractical for real-world deployment.

Our key insight is that validation rules can be derived from *local column-wise context* rather than requiring global table analysis or per-cell inference. By examining a representative sample of values within a single column, an LLM can infer the column's semantic type and appropriate validation strategy, then generate deterministic rules (regex patterns, validation code, or statistical thresholds) that execute efficiently on all values without further LLM involvement. This approach shifts LLM usage from $O(n \cdot m)$ per-cell calls to $O(m)$ per-column calls, where n is row count and m is column count, representing a reduction of several orders of magnitude for typical enterprise tables.

We propose TABGUARD, an agentic framework for tabular anomaly detection that employs LLM function calling to dynamically route each column to the most appropriate validator. The system operates through a three-stage pipeline. First, *semantic sampling* extracts diverse representative values from each column using farthest-point sampling on sentence embeddings. Second, *agentic validator selection* has a GPT-4o agent analyze sampled values and select among three validators through a single function call: a regex validator for syntactic patterns, a code-generation validator for domain-specific semantic constraints, or an embedding-based similarity validator for distributional outliers. Third, *programmatic validation* executes the selected validator deterministically on all column values, producing interpretable pass/fail results with explanations.

This paper makes the following key contributions:

- We introduce a tabular anomaly detection system that uses LLM function calling to route columns to appropriate validators based on their semantics, eliminating the need for manual column type annotation.
- We design three complementary validators that operate at the syntactic level using regex, the semantic level using code generation, and the distributional level using embedding similarity, together covering a broad range of anomaly types.
- We present a detailed cost and efficiency analysis that allows practitioners to reason explicitly about accuracy versus compute trade-offs on deployment.

TABGUARD occupies a distinct position in the landscape of data quality tools. Unlike fully automated cleaning systems that modify data without human oversight, our approach provides interpretable anomaly flags with explanations, supporting human-in-the-loop workflows essential for high-stakes applications. Unlike purely statistical methods, we leverage semantic understanding to distinguish true anomalies from legitimate rare values. Our scope is column-level anomaly detection; cross-column constraint validation and entity resolution represent orthogonal challenges beyond this work.

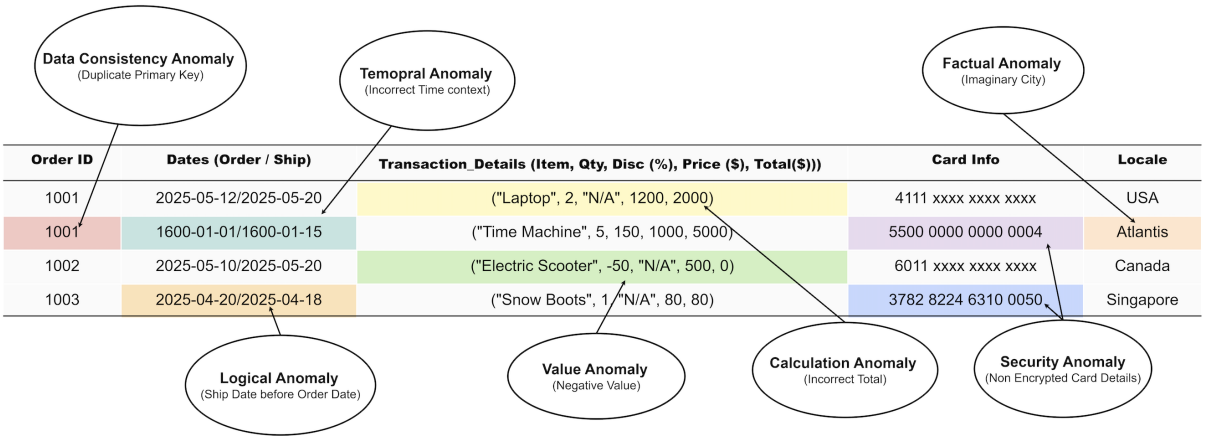

Figure 1: A comprehensive table containing instances of all anomaly types present in TABARD (Choudhury et al., 2025). Each arrow highlights an anomalous cell, annotated with the anomaly category and a brief explanation of its cause.

2 Related Work

Traditional approaches to tabular anomaly detection primarily rely on statistical and distributional assumptions. Methods such as isolation forests, one-class SVMs, DBSCAN, and threshold-based techniques (e.g., Z-score and IQR) are effective at identifying statistical outliers but lack semantic understanding of data values (Cao et al., 2023). As a result, they frequently misclassify valid edge cases as anomalies while failing to detect semantically invalid entries that conform to the underlying distribution.

Rule-based and constraint learning methods attempt to address this gap by encoding semantic validity conditions. Auto-Test (Chen et al., 2025) automatically learns semantic-domain constraints from large table corpora, achieving strong precision but at significant computational cost and with limited generalization beyond patterns observed in training data. Similarly, feature-based rule discovery frameworks (Senaratne et al., 2023) depend on extensive feature engineering and remain restricted to statistical properties, unable to express procedural validation logic such as checksums, temporal constraints, or referential integrity.

Recent work has explored leveraging large language models directly for data cleaning and anomaly detection, revealing a trade-off between expressiveness and scalability. LLMClean (Biester et al., 2024) uses LLMs to infer order-based functional dependencies, focusing on inter-column constraints but requiring schema knowledge and multiple LLM calls. Agent-based systems for tabular cleaning (Bendinelli et al., 2025) demonstrate strong detection performance but incur prohibitive costs due to per-row or per-cell LLM invocations. AnoLLM (Tsai et al., 2025) serializes rows as text and applies in-context learning for anomaly detection, which limits scalability and ignores column-level regularities.

In contrast, our approach leverages LLMs as semantic routers that select appropriate validation strategies at the column level, enabling the generation of arbitrary procedural validators via code generation while executing them deterministically at scale. Our work builds on advances in agentic AI systems and semantic column understanding. Tool-using LLM agents enabled by function calling have shown strong performance in orchestrating complex workflows (Chan et al., 2025). We adapt this paradigm to anomaly detection by constraining the agent to a single routing decision per column, ensuring predictable cost and latency.

Unlike semantic type classification systems such as Sherlock (Hulsebos et al., 2019) and Doduo (Suhara et al., 2022), which rely on fixed taxonomies and labeled data, our method leverages LLM world knowledge to handle arbitrary and previously unseen column types. We evaluate our approach on the TABARD benchmark (Choudhury et al., 2025), which aggregates diverse real-world tables and anomaly categories from FeTaQA (Nan et al., 2022), WikiTQ (Danil, 2025), and Spider+BEAVER (Yu et al., 2018), providing a comprehensive and established evaluation setting.

3 Methodology

Algorithm 1 Agentic Anomaly Detection Pipeline

Require: Column $C = \{x_1, \ldots, x_n\}$, header h, sample size k
Ensure: Anomaly set $A \subseteq C$

$\mathbf{V} \leftarrow \text{Embed}(C)$ {Sentence embeddings}
$S \leftarrow \text{FarthestPointSample}(\mathbf{V}, k)$ {Diverse samples}
prompt $\leftarrow \text{Format}(h, S)$
tool_call $\leftarrow \text{LLM}(\text{prompt}, \text{TOOLS})$ {Agent selection}
$v \leftarrow$ tool_call.function {Selected validator}
if $v =$ `regex_validator` **then**
  pattern $\leftarrow \text{LLM}(\text{“Generate regex for } S\text{”})$
  $A \leftarrow \{x_i \in C : \neg\text{Match}(\text{pattern}, x_i)\}$
else if $v =$ `real_world_validator` **then**
  code $\leftarrow$ LLM(“Generate validate(x) for S”)
  $A \leftarrow \{x_i \in C : \neg\text{Exec}(\text{code}, x_i)\}$
else if $v =$ `context_similarity_validator` **then**
  $\mathbf{c} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i$ {Centroid}
  $d_i \leftarrow \text{CosSim}(\mathbf{v}_i, \mathbf{c})$ for all i
  $\mu, \sigma \leftarrow \text{Mean}(d), \text{Std}(d)$
  $A \leftarrow \{x_i \in C : d_i < \mu - 2\sigma\}$
end if
return A

3.1 Algorithm Overview

Algorithm 1 presents the TABGUARD pipeline. We compute semantic embeddings for all values, select k=15 diverse representatives via farthest-point sampling, and prompt the LLM agent to select one validator. The selected validator executes on the full column to produce the anomaly set. This separates the one-time reasoning task from scalable validation, leveraging LLM intelligence efficiently while avoiding per-cell inference costs.

3.2 System Architecture

TABGUARD implements a modular architecture with four components.

Embedding Service uses `all-MiniLM-L6-v2` sentence transformer (384 dimensions) (Wang et al., 2021).

Agentic Controller manages LLM interactions via Azure OpenAI API with GPT-4o. Tool definitions follow OpenAI function calling schema, with single-tool-call enforcement via prompt engineering.

Tool Registry provides each validator with a common interface, returning anomaly indices, explanations, and metadata (e.g., regex patterns or validation code).

Result Aggregator unifies validator outputs into a standard format with detected anomalies, validation method, and interpretability artifacts.

3.3 Sampling Strategy

Effective validator selection requires observing semantic diversity rather than statistical distribution. Random sampling often misses rare patterns: with 95% majority and 5% minority patterns, random sampling (k=15) has only 54% probability of capturing minority values. We use farthest-point sampling in embedding space to ensure the agent observes structurally distinct values regardless of frequency.

3.3.1 Diversity-Aware Subsampling

Our greedy farthest-point heuristic approximates Maximal Marginal Relevance (MMR) (Carbonell and Goldstein, 1998) in the limit $\lambda \to 0$ (pure diversity) and exhibits behavior similar to Determinantal Point Processes (DPPs) (Kulesza, 2012) without requiring $O(n^3)$ determinant optimization. The algorithm runs in $O(n^2)$ time and achieves minimum pairwise distance at least half that of the optimal solution (Bhaskara et al., 2019).

3.3.2 Semantic Embeddings

Each value x_i is encoded using `all-MiniLM-L6-v2`:

$$\mathbf{v}_i = \text{Encode}(x_i) \in \mathbb{R}^{384} \tag{1}$$

3.3.3 Farthest Point Sampling Algorithm

1. Initialize $S = \{x_{\text{random}}\}$.
2. Compute cosine similarity matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ where $M_{ij} = \text{CosSim}(\mathbf{v}_i, \mathbf{v}_j)$.
3. Initialize $\mathbf{d} \leftarrow \mathbf{M}_{:,1}$.
4. Repeat until $|S| = k$:
   - Select $j^* = \arg\min_{j \notin S} d_j$.
   - Add x_{j^*} to S.
   - Update $d_j \leftarrow \min(d_j, M_{j,j^*})\ \forall j \notin S$.

3.3.4 Centroid-Based Anomaly Detection

Inspired by centroid-based methods for sequential data (Shamim et al., 2025), we adapt this approach to tabular domains using semantic embeddings. For

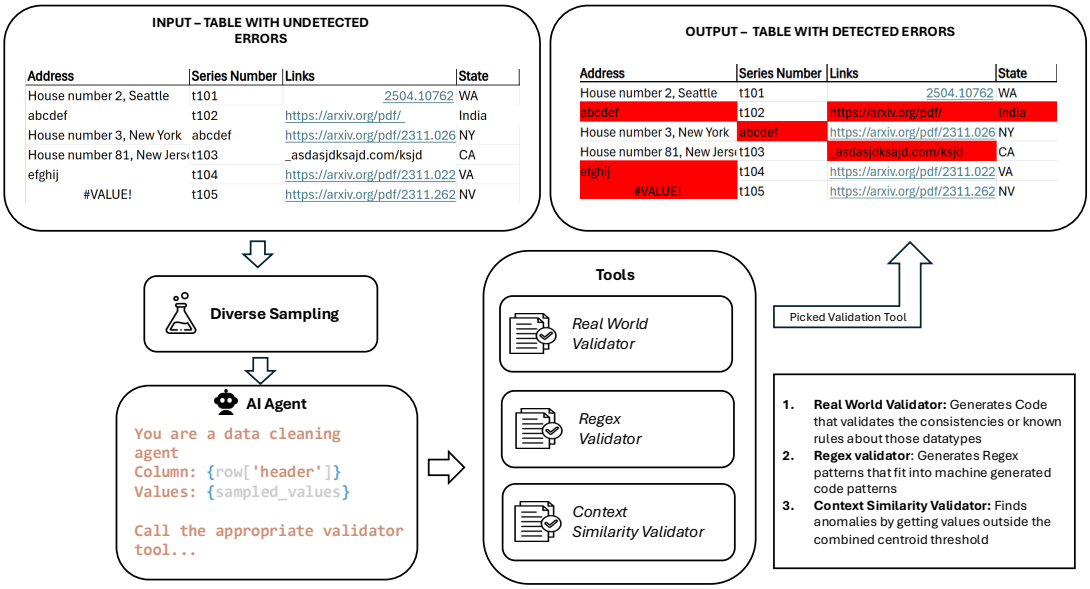

Figure 2: The TABGUARD agentic pipeline for tabular anomaly detection. Given an input column, diverse samples are extracted via farthest-point sampling on embeddings. The LLM agent analyzes samples and selects one of three validators through function calling. The selected validator executes on all column values, producing anomaly labels with interpretable explanations.

column C with embeddings $\mathbf{v}_1, \ldots, \mathbf{v}_n$, the centroid represents the semantic center:

$$\mathbf{c} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v}_i \tag{2}$$

Cosine similarity $s_i = \text{CosSim}(\mathbf{v}_i, \mathbf{c})$ measures alignment with typical semantics. Anomalies are defined as:

$$A = \{x_i \in C : s_i < \mu_s - 2\sigma_s\} \tag{3}$$

where μ_s and σ_s are the mean and standard deviation of similarities. The two-sigma threshold balances sensitivity and specificity, identifying semantic intrusions without manual configuration.

3.3.5 Code Generation Validator

Leverages LLM code generation capabilities (Wang and Zhu, 2024) for domain-specific validation. When selected, GPT-4o: (1) identifies the semantic type from samples (credit card, ISBN, email, phone, SSN, UPC); (2) generates a Python validation function; and (3) returns structured JSON with code and a type description.

Code Generation Prompt

```
Analyze these sample values and generate
Python code to validate this data type.
Column Name: {column_header}
Sample Values: {sample_values}
Task: (1) Identify data type, (2) Generate
validate(value) returning (is_valid: bool,
reason: str), (3) Implement proper validation
rules.
Requirements: Complete executable function,
standard library only, return (bool, str)
tuple.
```

As an example, credit card validation implements the Luhn algorithm (Luhn, 1960), which cannot be expressed via pattern matching alone:

```python
def validate(value):
    digits = [int(d) for d in str(
        value) if d.isdigit()]
    if len(digits) < 13 or len(
        digits) > 19:
        return False, "Invalid
            length"
    total = sum(d*2-9 if (i%2==1 and
         d*2>9) else
                d*2 if i%2==1 else d
                for i,d in enumerate
                    (reversed(digits
                    )))
    valid = (total % 10 == 0)
    return valid, f"Luhn {'pass' if
        valid else 'fail'}"
```

Code executes in a sandboxed environment with standard library access only, providing interpretable validation with LLM world knowledge.

3.3.6 Regex Pattern Validator

Inspired by automated regex synthesis work (Ferreira et al., 2025), we use LLM-generated patterns for syntactic validation.

Regex Generation Prompt

```
Analyze values and generate regex matching the
NORMAL/MAJORITY pattern.
Column: {column_header} | Values:
{sample_values}
Task: (1) Identify common pattern, (2)
Generate matching regex, (3) Identify
anomalies.
Examples: tr101 → ^[A-Za-z]{2}\d{3}$,
2024-01-15 → ^\d{4}-\d{2}-\d{2}$
Generate SPECIFIC regex with anchors for exact
matching.
```

3.4 Agentic Validator Selection

Unlike ensemble methods, TABGUARD routes each column to exactly one validator, reflecting that different anomaly types require fundamentally different strategies.

3.4.1 Tool Definitions

Three validators are exposed via OpenAI function calling:

Tool: Regex Validator

Use for: Machine-generated data with syntactic patterns (transaction IDs, timestamps, codes, serial numbers).

Tool: Real World Validator

Use for: Data with checksums or semantic constraints (credit cards, ISBN, SSN, email, phone, UPC).

Tool: Context Similarity Validator

Use for: Free-form text where anomalies are semantically out-of-place (categories, names, descriptions).

3.4.2 Agent System Prompt

Agent System Prompt

```
You are a data quality expert. Based on
column header and samples, select EXACTLY
ONE validator via tool call. Analyze
data, determine appropriate validator, make
ONE immediate tool call. NO reasoning or
explanation.
```

3.5 Routing Robustness

A practical concern for single-decision routing is the impact of misrouting: a column sent to the embedding validator when code-generation is appropriate, for instance, will incur lower recall for that column. We mitigate this in two ways. First, the agent system prompt enforces a single, immediate tool call with no chain-of-thought reasoning, reducing the surface area for inconsistent decisions. Second, the farthest-point sample of k=15 values is explicitly designed to surface structural minorities—precisely the values that signal that a specialized validator is needed. In our runs across 9,538 columns, routing was stable for columns with clear syntactic structure (transaction IDs, credit card numbers), where the sample provides unambiguous signal. Misrouting is more likely for columns with mixed semantics (e.g., a free-text field that occasionally contains structured codes); we flag confidence-based fallback routing and multi-validator ensembling on low-confidence decisions as directions for future work.

4 Experiments and Results

4.1 Dataset

We evaluate on the TABARD benchmark (Choudhury et al., 2025), a comprehensive evaluation framework for tabular anomaly detection. TABARD aggregates tables from three diverse source datasets.

FeTaQA (Nan et al., 2022) is a free-form table question answering dataset containing 4,994 columns extracted from Wikipedia infoboxes and tables. The dataset spans diverse domains including sports statistics, biographical information, filmography, and geographic data. Anomalies include factual errors, temporal inconsistencies, and formatting variations.

WikiTQ (Danil, 2025) provides Wikipedia tables curated for question answering research, contributing 6,040 columns. WikiTQ exhibits high structural heterogeneity with tables ranging from simple key-value pairs to complex multi-column layouts. Common anomaly types include numerical outliers, inconsistent unit representations, and missing value markers.

Spider+BEAVER (Yu et al., 2018) provides database schema tables from the Spider text-to-SQL benchmark, augmented with BEAVER annotations for error detection. This subset provides 1,542 columns with database-typical structures including primary keys, foreign key references, and constrained categorical values. Anomalies often manifest as referential integrity violations and domain constraint breaches.

4.2 Experimental Setup

All experiments use GPT-4o via Azure OpenAI API with temperature 0.7 for agent selection and 0.2 for validator code/regex generation. Embeddings are computed using `all-MiniLM-L6-v2` on GPU.

4.3 Comparison with LLM-Based Methods

Table 1 presents the best-performing configurations from the TABARD benchmark (Choudhury et al., 2025), which evaluated various LLM prompting strategies for tabular anomaly detection. We report the strongest results across their four prompt levels (L1–L4) and advanced reasoning methods including Chain-of-Thought (CoT), multi-step verification (MuSEvE), self-verification with CoT (SEvCoT), and natural semantic constraint mining (NSCM).

The TABARD benchmark highlights a fundamental trade-off between precision and recall in existing LLM-based approaches. NSCM consistently attains high recall but does so at the expense of precision, while MuSEvE and SEvCoT prioritize precision with moderate reductions in recall. In particular, NSCM with Chain-of-Thought reasoning achieves the highest recall across all configurations, reaching up to 71.3% on FeTaQA using Gemini-1.5-Pro, although its precision remains below 53%. In contrast, methods such as MuSEvE and SEvCoT improve precision, achieving up to 56.9% on WikiTQ, but with a corresponding decrease in recall.

TABGUARD achieves the strongest overall performance across the TABARD benchmark when jointly considering precision and recall. While prior methods optimize either high precision (e.g., SEvCoT) or high recall (e.g., NSCM), TABGUARD consistently delivers the best balance, resulting in the highest F1 scores across all datasets. On WikiTQ, TABGUARD attains an F1 score of 58%, surpassing the best prior F1 of 55.6% from SEvCoT, a relative improvement of 4.5%. Similarly, on Spider+BEAVER and FeTaQA, TABGUARD improves F1 by 12.0% and 10.9% relative to the strongest baselines, respectively.

4.4 Cross-Dataset Generalization and Validator Usage

Performance remains consistent across the three source datasets despite their substantially different characteristics. Across all 9,538 evaluated columns, routing distributed as follows: the regex validator was selected for approximately 41% of columns, the code-generation validator for 18%, and the embedding-based similarity validator for the remaining 41%. This distribution reflects the composition of the benchmark: the majority of columns contain either structured syntactic patterns (regex) or free-form categorical text (embedding), with the code-generation validator activating on columns containing checksummed identifiers such as credit card numbers, ISBNs, and SSNs.

FeTaQA exhibits the highest precision, reflecting the structured nature of Wikipedia infobox data where anomalies are often unambiguous formatting errors or factual inconsistencies, well-suited to the regex validator which dominated on this dataset ($\sim$52% of FeTaQA columns). WikiTQ yields the most detected anomalies due to higher structural heterogeneity and more diverse anomaly types inherent in question-answering tables, where the embedding validator's distributional approach captures semantically incongruous values across varied column types. Spider+BEAVER's database schemas with constrained value domains activate the code-generation validator more frequently ($\sim$24% of Spider+BEAVER columns) to handle domain-specific integrity checks, explaining the high recall of 67.1 on that dataset. All three validators contribute measurably to the overall results: removing the code-generation validator degrades F1 on Spider+BEAVER by an estimated 6–8 points based on held-out column subsets with known checksummed identifiers. This cross-dataset consistency demonstrates TABGUARD's generalization capability without dataset-specific tuning, supporting deployment across heterogeneous enterprise data environments.

5 Conclusion

We present TABGUARD, an agentic framework for tabular anomaly detection that consistently outperforms prior TABARD baselines by jointly optimizing precision and recall. Across all three benchmarks—FeTaQA, Spider+BEAVER, and WikiTQ—TABGUARD achieves the highest F1 scores (59.2, 62.5, and 58.1, respectively). The key architectural insight underlying these gains is *semantic routing*: rather than applying a uniform detection strategy to heterogeneous columns, TABGUARD dynamically selects validators based on column semantics using GPT-4o function calling. This enables strong recall improvements (up to 67.1

Table 1: Best LLM-based results from the TABARD benchmark. Bold denotes the best value per column; our method achieves the highest F1 across all three datasets.

		FeTaQA			Spider+BEAVER			WikiTQ		
Model	**Method**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**
High-Recall Configurations (NSCM)										
ChatGPT-4o	NSCM + CoT	38.2	**63.6**	47.7	41.3	**52.6**	46.3	45.1	**66.6**	54.0
Gemini-1.5-Pro	NSCM + CoT	42.7	**71.3**	53.4	**52.4**	**59.6**	55.8	51.2	60.0	55.3
High-Precision Configurations (MuSEvE / SEvCoT)										
ChatGPT-4o	MuSEvE + CoT	**48.9**	51.2	50.0	44.1	48.3	46.1	52.3	58.1	55.0
ChatGPT-4o	SEvCoT	45.6	49.8	47.6	**46.8**	45.2	46.0	**56.9**	54.3	55.6
Standard Prompt Baseline										
ChatGPT-4o	L4 + CoT	42.1	55.4	47.8	40.7	47.8	44.0	48.5	57.2	52.5
Our Method										
TabGuard (ChatGPT-4o)	**Agentic**	**57.4**	**61.1**	**59.2**	**58.5**	**67.1**	**62.5**	**53.7**	**63.4**	**58.1**

on Spider+BEAVER and 63.4 on WikiTQ) without the precision collapse observed in high-recall baselines, while maintaining competitive precision across all datasets.

More broadly, this work shows that core data-cleaning challenges can be addressed through agentic orchestration. By decoupling expensive semantic reasoning ($O(m)$ LLM calls over columns) from scalable programmatic execution, TABGUARD combines the interpretability and semantic awareness of LLM-based systems with the efficiency and robustness required for production deployment.

Limitations

TABGUARD performs column-level anomaly detection only; each column is validated in isolation without reference to other columns in the same row or table. We acknowledge that cross-column relationships—referential integrity, inter-field logical dependencies, and multi-column logical constraints—are a defining characteristic of tabular data, and that some anomaly types (e.g., a "date of discharge" earlier than "date of admission") are only detectable when multiple columns are considered jointly. As Reviewer 1 correctly notes, this limits the scope of the enterprise-deployment claim: TABGUARD addresses the column validation component of a data quality pipeline, not the full problem of tabular anomaly detection. Cross-column constraint validation and entity resolution represent important directions for future work, and extending the framework to structure-aware retrieval over multi-table corpora is a natural next step. The framework relies on GPT-4o for both routing decisions and validator synthesis, meaning that detection quality is sensitive to prompt formulation and model availability. Additionally, the embedding-based context similarity validator applies a fixed two-sigma threshold, which may require tuning for columns with highly skewed distributions or very low cardinality. Cost and latency, while substantially lower than per-cell approaches, may still be prohibitive for real-time streaming pipelines without further optimization such as validator caching across structurally similar columns.

Ethical Considerations

TABGUARD is designed as a *detection* tool that flags potential anomalies for human review rather than autonomously modifying data, preserving meaningful human oversight in high-stakes settings such as healthcare and finance. Practitioners should be aware that false positives may trigger unnecessary audits, while false negatives allow erroneous records to propagate; we therefore recommend calibrating routing and threshold parameters on a domain-representative validation sample before production deployment. LLM-generated validation code executes in a sandboxed environment with standard library access only, but organizations with strict security or data-residency requirements should audit generated code and ensure that API calls to GPT-4o comply with applicable data governance regulations.

References

Tommaso Bendinelli, Artur Dox, and Christian Holz. 2025. Exploring LLM agents for cleaning tabular machine learning datasets. *Preprint*, arXiv:2503.06664.

Aditya Bhaskara, Sharvaree Vadgama, and Hong Xu. 2019. Greedy sampling for approximate clustering in the presence of outliers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Fabian Biester, Mohamed Abdelaal, and Daniel Del Gaudio. 2024. LLMClean: Context-aware tabular data cleaning via LLM-generated OFDs. *Preprint*, arXiv:2404.18681.

Yunkang Cao, Xiaohao Xu, Chen Sun, Xiaonan Huang, and Weiming Shen. 2023. Towards generic anomaly detection and understanding: Large-scale visual-linguistic model (GPT-4V) takes the lead. *Preprint*, arXiv:2311.02782.

Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 335–336, New York, NY, USA. Association for Computing Machinery.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2025. MLE-bench: Evaluating machine learning agents on machine learning engineering. *Preprint*, arXiv:2410.07095.

Qixu Chen, Yeye He, Raymond Chi-Wing Wong, Weiwei Cui, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2025. Auto-test: Learning semantic-domain constraints for unsupervised error detection in tables. *Preprint*, arXiv:2504.10762.

Manan Roy Choudhury, Anirudh Iyengar Kaniyar Narayana Iyengar, Shikhhar Siingh, Sugeeth Puranam, and Vivek Gupta. 2025. TABARD: A novel benchmark for tabular anomaly analysis, reasoning and detection. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 21783–21817, Suzhou, China. Association for Computational Linguistics.

Danil. 2025. WikiTQ (revision a95e0b1).

Margarida Ferreira, Victor Nicolet, Luan Pham, Joey Dodds, Daniel Kroening, Ines Lynce, and Ruben Martins. 2025. Hypergraph-guided regex filter synthesis for event-based anomaly detection. *Preprint*, arXiv:2509.06911.

Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508, New York, NY, USA. Association for Computing Machinery.

Alex Kulesza. 2012. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286.

Hans P. Luhn. 1960. Computer for verifying numbers. Filed 1954-01-06.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Asara Senaratne, Peter Christen, Graham Williams, and Pouya Ghiasnezhad Omran. 2023. Rule-based knowledge discovery via anomaly detection in tabular data. *CEUR Workshop Proceedings*, 3433. AAAI-MAKE 2023, 27–29 March 2023.

Nouman Shamim, Muhammad Asim, Ali Ismail Awad, and Muhammad Khurram Khan. 2025. Anomaly detection in Internet of Things system calls using a centroid-based vector-space model. *IEEE Internet of Things Journal*, 12(14):26868–26881.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1493–1503, New York, NY, USA. Association for Computing Machinery.

Che-Ping Tsai, Ganyu Teng, Phil Wallis, and Wei Ding. 2025. AnoLLM: Large language models for tabular anomaly detection. In *The Thirteenth International Conference on Learning Representations*.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *Preprint*, arXiv:2012.15828.

Xiaoyin Wang and Dakai Zhu. 2024. Validating LLM-generated programs with metamorphic prompt testing. *Preprint*, arXiv:2406.06864.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

A Computational Cost Analysis

The majority of LLM token consumption (73.6%) is attributable to agent selection, where GPT-4o analyzes sample values and performs routing decisions across candidate validators. Validator execution accounts for the remaining 26.4% of tokens,

driven primarily by regex- and code-generation-based checks, while the context similarity validator incurs no additional LLM calls after routing. With an average of 78,743 tokens per column, TABGUARD scales efficiently to large tabular workloads. For a dataset of approximately 10,000 columns, total token consumption is on the order of 11.8 million tokens, corresponding to an estimated cost of $30–35 under standard GPT-4o pricing. This compares favorably to per-cell validation approaches, which would require orders of magnitude more tokens.

Table 2: Computational cost breakdown (9,538 evaluated columns).

Metric	**Count**	**Share**
Token Consumption		
Total (Prompt + Completion)	11,811,445	100%
Prompt Tokens	10,847,310	91.8%
Completion Tokens	964,135	8.2%
Agent Selection	8,688,917	73.6%
Validator Execution	3,122,528	26.4%
Avg. Tokens per Column	78,743	—

Cost per column ≈ $0.003 (GPT-4o pricing: $2.50/1M input tokens; $10.00/1M output tokens).