Can LLMs Serve as a Data Error Detection Engine? Trade-offs in Accuracy, Cost, and Hallucination Across Datasets

Maximilian Plazotta
University of Regensburg
Regensburg, Germany
maximilian.plazotta@informatik.uni-regensburg.de

Meike Klettke
University of Regensburg
Regensburg, Germany
meike.klettke@informatik.uni-regensburg.de

ABSTRACT

Large Language Models (LLMs) are emerging as important tools in the field of data, with error detection being a particularly promising application area. Conventional data error detection techniques often rely on domain-specific knowledge or multi-step processes, restricting their adaptability and scalability. Conversely, LLMs incorporate vast amounts of domain knowledge, having undergone pre-training on billions of data points, and can be easily deployed as an inference engine, too. This study evaluates the effectiveness of LLMs in detecting errors across datasets of different sizes and complexities, employing a novel LLM-based error detection framework. Ten LLMs, comprising four open-source and six closed-source models, are assessed based on the parameters of accuracy and execution time. The results indicate that accuracy decreases and execution time increases as dataset sizes and error rates increase. Proprietary models surpass open-source models, and larger closed-source models yield superior outcomes compared to their smaller equivalents, contradicting original assumptions. These findings endorse the viability of LLMs as a feasible alternative for data error detection; nonetheless, challenges such as hallucination and cost aspects remain, especially regarding the resources necessary for the effective implementation of these models in practical applications.

VLDB Workshop Reference Format:
Maximilian Plazotta and Meike Klettke. Can LLMs Serve as a Data Error Detection Engine? Trade-offs in Accuracy, Cost, and Hallucination Across Datasets. VLDB 2026 Workshop: 15th International Workshop on Quality in Databases (QDB'26).

1 INTRODUCTION

LLMs are increasingly used to solve data-centric tasks, resulting in significant progress in domains such as text-to-SQL, schema matching, internal database components cardinality estimation, and query optimization. One of the most prominent applications resides in data error detection, where LLMs can assist in identifying anomalies, misplaced values, or contextual errors in datasets. Traditional methods, such as using knowledge bases (KB), ontology look-ups, or embedding-based techniques, have inherent limitations arising from their dependence on specialized domain knowledge and the need for several separate steps. These limitations do not apply to LLMs, as they are trained on billions of pieces of domain data, incorporating vast amounts of domain knowledge, and they are easy to integrate. The problem statement is rooted in the question of whether LLMs are already capable of serving as a data error detection engine. With the increasing volume, velocity, and complexity of data, such a system must perform accurately, focusing on scalability and timeliness while also being trustworthy. In this regard, our experiments aim to test exactly these parameters through the following four **hypotheses (H)**:

- **H1**: Closed-source models will achieve higher accuracy than open-source models (H1-1). Among the closed-source models, smaller models will outperform bigger models on well-defined tasks, following LLM providers' claims (H1-2).
- **H2**: An increase in attribute cardinality – the number of distinct values of an attribute – will yield lower accuracy.
- **H3**: Expanding dataset sizes will reduce accuracy and increase execution time.
- **H4**: The more errors (i.e., higher error rates) there are in a dataset, the lower the accuracy and execution time will be.

Our contribution to the existing body of research is an LLM-based error detection framework for evaluating the effectiveness of LLMs in finding data errors across varying dataset sizes, error rates, and domain types. Our approach targets the detection phase within the data quality (DQ) lifecycle [16], specifically identifying contextual (string) errors. Therefore, the present work can be positioned as a component within, rather than a replacement for, the full DQ lifecycle. The framework measures accuracy and execution time for ten different LLMs across 48 datasets from four domains. Our findings reveal that accuracy and execution time decline when dataset sizes and error rates increase. Moreover, private models outperform their open-source equivalents significantly. Our work promotes the use of LLMs as a data error detection engine and provides a realistic alternative to conventional approaches.

2 BACKGROUND AND RELATED WORK

A plethora of methods exist for detecting out-of-place, contextual errors in string datasets. We discuss the four most common ones and conclude this section with the "new" LLM-oriented approach.

2.1 Statistical String Analysis

The first method, **Statistical String Analysis**, analyzes the distribution of string values within a text document to find potential errors and flag them for further examination. The fundamental assumption is that errors within a dataset follow a specific pattern and

Proceedings of the VLDB Endowment. ISSN 2150-8097.

accumulate. The widely used approach "Term Frequency-Inverse Document Frequency" (TF-IDF) – introduced in 1988 [15] – measures how distinct individual words are from the entire corpus and assigns them weights. However, TF-IDF, despite its simplicity and computational efficiency, has a severe drawback: the assumption of independence among terms ignores semantic dependencies or context, which is not applicable to most real-world datasets.

2.2 Knowledge Base and Ontology Look-up

Knowledge Base (KB) and Ontology Look-up provide a structured and semantically coherent way to store domain knowledge. The principle behind this method is straightforward: data points are validated against the repository for existence, range, and relations between entities. A commonly used KB on general-purpose knowledge is DBpedia, which captures information from Wikipedia. For specialized, domain-specific ontologies, the WordNet database – a lexical ontology for English words – represents another popular example. In addition, there exist more sophisticated frameworks built on top of KBs: SPARQL [13] and OWL (Web Ontology Language) [7] must be mentioned here. Nevertheless, these tools share one major problem: **the knowledge is limited by the repository**.

2.3 Embedding-based Semantic Similarity Approach

(Partially) approaching LLM territory, the **Embedding-based Semantic Similarity Approach** is a powerful and widely used method for finding data errors. At its core, it measures the similarity between words or phrases based on their semantic meaning. These relationships are represented as numerical vectors in a high-dimensional space – referred to as embeddings. As a result, semantically similar words are located close to each other in the vector space. Frequently used embeddings include **GloVe, Word2Vec, and BERT**, which differ significantly; Word2Vec [10, 11] and GloVe [12] are static word embeddings and are simpler in their architecture. Conversely, BERT [5] is defined as a pre-trained language model (PLM) and relies on a transformer-based architecture, as LLMs do. BERT and LLMs differ in the transformer architecture: BERT is an encoder-only model, whereas LLMs are decoder-only models. The generated embeddings can subsequently serve as input for downstream tasks such as text classification, clustering, or anomaly detection. This approach involves many individual process steps and is logically more time-consuming than using an LLM directly.

2.4 LLMs for Data Error Detection

To reduce complexity by eliminating the process steps described in Section 2.3, why not use LLMs directly? Furthermore, LLMs are pre-trained on vast amounts of data (sources) that exceed the depth of any single KB, ontology, or embedding, yielding a superior "knowledge base". A typical pipeline incorporates the data into the prompt in a machine-readable format such as `.json` or, for tabular data, `.csv` or markdown tables. More sophisticated architectures use the recently developed **Model Context Protocol (MCP)** servers to connect to predefined resources (e.g., databases such as Postgres). However, this new approach is restricted by the high cost of transmitting large amounts of data to LLMs. Hallucination represents another problem that can lead to incorrect or heavily biased responses. Thus, although LLMs may appear as powerful tools in this area, their output must always be challenged and checked. This notion is especially relevant given that leveraging **LLMs for data error detection** has garnered considerable attention, especially in 2024 and 2025. Ren et al. (2025) [14] present an extensive table of scientific papers using LLMs for anomaly detection or prediction tasks. Two main observations emerge: tabular and time-series data are underrepresented (10–15%), and the majority of foundation models used are OpenAI's GPT and/or Llama models. Notably, the second aspect is evident in later academic studies; for instance, Li et al. (2024) [9] demonstrated that LLMs can operate as zero-shot, batch-level error detectors for tabular data, using GPT-3.5, GPT-4, Llama2-7B, and Mistral-7B. Glock et al. (2025) [6] assessed the effectiveness of LLMs in detecting errors in personal contact information and discovered that they significantly outperform existing approaches – GPT-4.1 and Llama-3 performed much better compared to Llama-4 and DeepSeek-R1. Despite these advancements, significant limitations remain, as shown in the case study of Chernigovskaya et al. (2025) [3]. Their research indicates that open-source LLMs struggled with numerical data processing and data analysis tasks in the manufacturing sector. Yang et al. (2025) [17] introduced a new LLM benchmarking framework, AD-LLM, to assess the efficacy of LLMs on NLP anomaly detection tasks. Their findings demonstrate that the chosen LLMs – OpenAI-o1-preview, OpenAI-o3-mini, and DeepSeek-R1 – excel in zero-shot scenarios.

2.5 Multi-table, multi-column Data Error Detection

The detection and correction of errors in **multi-table, multi-column** datasets have been one of the main research directions recently. Abedjan et al. [1] evaluated several tools to identify data errors and found that no single tool can correct all errors at the same time. They found that the order in which the tools are applied heavily impacts the results. To tackle this optimization problem, they proposed an automated, multi-tool framework with interactive user guidance. Another important study in this field focuses on the fragmentation of data among many tables: Ahmadi et al. [2] developed Matelda, a system that uses automated error detection with a human-in-the-loop approach in a multi-table setup. Matelda allows users to improve the system interactively based on their own expert knowledge.

The above-mentioned investigations show the potential and limitations of LLMs for error detection. However, the majority of the studies mainly focused on the use of a relatively small number of models, where OpenAI's GPT and Llama are the most commonly used. Moreover, the detection of data errors in tabular datasets is not sufficiently addressed in the existing literature. This paper seeks to contribute to both of these shortcomings.

3 METHODOLOGY

The aim of this study is to empirically evaluate whether LLMs can serve as a straightforward data error detection engine. This method stems from the underlying assumption that LLMs provide a superior understanding of semantic patterns and contextual relations coded within them through their vast amount of diverse training data.

3.1 LLM-based Error Detection Framework

To test our hypotheses, we derived a framework that applies LLMs to scan for logical and contextual anomalies across datasets. The framework functions as a pipeline with five central components: **dataset generation, data preprocessing, prompt construction (including data embedding), API communication, and response analysis** which will be explained in detail in Section 4.1, 'Experimental Setup'. To assess the effectiveness of the LLM-based error detection framework, we apply standard metrics of accuracy and execution time to evaluate the performance of each model. Within the accuracy metric, we also analyze the negatives (not-found values) to assess why certain expressions are not detected by the model. The accuracy metric in this paper is defined as follows:

$$\text{accuracy} = \frac{\text{detected errors}}{\text{total errors}}$$

and basically reflects the well-established recall metric in the data quality domain. During each run, the time was measured to be able to evaluate the system's efficiency. In general, the framework is constructed to be modular, allowing different LLMs and datasets to be incorporated easily with minimal adjustments.

3.2 Prompt Design for Error Detection

Precise and clear prompting is essential so that the model's response fulfills the request satisfactorily. Under this notion, the term **prompt engineering** aims to provide the entire prompt with a structure that enables the LLM to respond in the best way possible. Figure 1 shows the prompts (Levels 1 and 2) used in our experiments, which feature a concise structure with three central components. The first block (lines 1-3) sets the tone: 'expert AI' gives the model an authoritative/competent persona, biasing it toward an analytical and logical tone instead of the normal conversational focus it inherits by default. This technique is called **role prompting**. Lines 4-6 provide the model with the input to analyze with a newline separator before and after. This is followed by the **explicit task decomposition** (lines 7-13), which breaks down the instructions into smaller, listed sub-tasks. Especially, lines 10-11 must be highlighted here, as they define the output format of the response. For Level 1 it only returns the faulty values; for Level 2 it is necessary to return the customer_id and the affected column to more easily locate the errors.

<Level 1 Prompt>

```
1  You are an expert AI designed to
2  detect out-of-place values in the
3  following list:
4
5    {values.csv}
6
7  Your task:
8  - Identify values that do NOT belong
9    with the rest.
10 - If any exist, return ONLY the faulty
11   values in a NEWLINE-SEPARATED list.
12 - If there are no faulty values, only
13   return: none.
```

<Level 2 Prompt>

```
1  You are an expert AI designed to
2  detect out-of-place values from the
3  following customer dataset:
4
5    {values.csv}
6
7  Your task:
8  - Identify values that do NOT belong
9    in the dataset.
10 - Focus on logical errors across
11   columns.
12 - If any errors exist, return the
13   customer_id and the affected column
14   like customer_id,column.
15 - If there are no faulty values, only
16   return: none.
```

Figure 1: Prompting templates for level 1 and level 2 datasets including labels for additional explanations.

3.3 Token-efficient Compression

Inference is not for free – both for closed-source and open-source models. For open-source models, the cost is negligible. When sufficient computational resources are available on a local machine, expenses are primarily driven by electricity consumption. In the absence of sufficient hardware, cloud services provide a remedy, albeit with pay-as-you-go pricing. Closed-source models have a different approach: they charge per token processed (input, cached, and output). Currently, the input price per one million tokens ranges between $2 and $5 for the flagship models, whereas their smaller counterparts are priced between $0.10 and $1. Output tokens are typically more expensive than input tokens. "One token is equivalent to one word" is a sentence often cited in discussions about token pricing and API cost. While this rule may hold for short words like 'tree' or 'house', longer words like 'scientist' (2 tokens) or 'interoperability' (3 tokens) require more tokens depending on the tokenizer [1]. To summarize, inference may appear cheap at first glance for one million tokens; however, when sending large volumes and/or at high frequencies of data to an API, the resulting costs accumulate quickly. There are ways to optimize token input, though. Basically, this involves removing all unnecessary elements without losing information. For example, in `.csv` files, data points are often enclosed in quotation marks ('value'), which creates additional tokens without adding semantic information. For flat tabular data, the `.csv` format with a comma as a delimiter is token-efficient. As we only use flat tabular data in our experiments, `.csv` format is sufficient and does not create additional preprocessing cost. How does this compare to other formats, such as `.json`? JSON is verbose, possesses many repetitive keys, and is

[1] https://platform.openai.com/tokenizer

punctuation-heavy (commas, braces, brackets, etc.), therefore being relatively token-expensive. A recently published GitHub project [8] called Token-Oriented Object Notation (TOON) addresses these inefficiencies, and the results are promising – 40% fewer tokens are used with better or equivalent retrieval accuracy.

4 EXPERIMENTS

To empirically validate the effectiveness of LLMs in detecting errors, we developed an extensive experimental setup based on our LLM-based error detection framework. We systematically varied dataset size, error rate, and domain types to test ten LLMs using accuracy and execution time as metrics.

4.1 Experimental Setup

In general, the pipeline involves the following five steps: **dataset generation, data preprocessing, prompt construction, API communication, and response analysis** (see Section 3.1). Overall, we constructed **48 datasets**, with 36 for Level 1 and 12 for Level 2 tests. In **Step 1**, we created a core list for the Level 1 datasets with our domain values and randomly extrapolated them based on the dataset size. Afterwards, we injected them with randomly, artificially created, meaningful, and non-domain-related words. For the Level 2 datasets, we created a synthetic customer dataset and corrupted it manually, following the guidelines from Section 4.2. **Step 2** ensures that the values propagated from our datasets to the LLM are token-efficient. The prompt construction (**Step 3**), being arguably the most crucial element within this pipeline, uses key aspects from the discipline of prompt engineering, which was extensively discussed in Section 3.2. The connection to the LLMs was established through their respective API endpoints (**Step 4**). For the open-source setup, we used Ollama for pulling the models – an NVIDIA GeForce RTX 5090 (32 GB of VRAM) plus 64 GB of DDR5 RAM was used for running the models (locally). The closed-source LLMs were accessed directly through API keys. The parameter temperature was set to 0, guaranteeing deterministic outputs, and token length was set to the maximum. The formatted data and the prompt were submitted to the LLM using zero-shot prompting; response times were logged for each query. As we defined within the prompt how the output should be structured, the LLM's response is then easily parsed and analyzed (**Step 5**) to compare errors (and also `false positives`) against the ground-truth errors.

4.2 Dataset Construction

LLMs are trained on billions of textual data points from various sources such as Wikipedia, Project Gutenberg, Reddit, etc. As a consequence, these models encode an extensive implicit knowledge base spanning a wide range of domains. In our experiments, we selected four different domains: **Level 1** includes **colors** with five different values (green, red, yellow, blue, and orange), **car brands** with 50 different values, and **capital cities** with 100 different observations. In the next step, different dataset sizes (100, 1,000, 10,000, 100,000) are created with a random distribution of values within a domain. Finally, random error words (e.g., tree, sky, dinner, etc.) are added – 1%, 10%, and 25% of the dataset size. For **Level 2**, the approach for the **customer dataset** differs slightly. The customer table consists of ten columns: customer_id, first_name, last_name, street, city, country, phone_number, birth_date, age, demographic_cohort. The dataset sizes are 1,000, 10,000, 20,000, and 40,000; e.g., a dataset of 1,000 contains 100 customers with 10 attributes, yielding 1,000 data points. The error rates are lower compared to Level 1, at 0.1%, 1%, and 10%. Here, we focus on functional dependency (FD) violations in a single customer record:

- city → street: The street is not located in the city.
- country → city: The city is not located in the country.
- country → phone_number: The phone number prefix mismatches the country's area code (e.g., +49 for Germany).
- birth date → age: The wrong age is calculated.
- age → demographic_cohort. Based on the age, the customer is wrongly grouped to the demographic cohort.

4.3 LLM Selection

Overall, we tested ten different LLMs listed in Table 1 – four open-source and six closed-source models. Within the six paid models, we selected one small and one large model from each vendor among their current state-of-the-art (SOTA) offerings. Most vendors claim that their smaller, cheaper models excel specifically in "well-defined tasks" [4] and in "(sub-) agentic tasks" [5] that our experiments reflect. From Anthropic, we chose Claude Haiku 4.5 (small) and Opus 4.5 (large); Google's Gemini models are represented by Gemini 3 Flash (small) and Gemini 3 Pro (large); and OpenAI is represented by GPT 5 Mini (small) and GPT 5.1 (large). From the open-source models, we selected gemma-3-27b-it, gpt-oss-20b, llama-3.3-70b-instruct, and olmo-3.1-32b-instruct. The reason for the selection is based on their advanced capabilities captured by the ELO score [4]. Furthermore, we highlight the cost and the respective context window (CW).

Table 1: LLM systems examined in this study

Open-source LLMs					
Model	Family	Provider	ELO[2]	Cost[3]	CW
gemma-3-27b-it	Gemma	Google	1365	-	128k
gpt-oss-20b	GPT	OpenAI	1317	-	128k
llama-3.3-70b-instruct	Llama	Meta AI	1319	-	128k
olmo-3.1-32b-instruct	Olmo	Allen AI	1330	-	64k
Closed-source (proprietary) LLMs					
Model	Family	Provider	ELO	Cost	CW
claude-haiku-4-5-20251001	Claude	Anthropic	1405	1$ / 5$	200k
claude-opus-4-5-20251101	Claude	Anthropic	1467	5$ / 25$	200k
gemini-3-flash	Gemini	Google	1473	0.5$ / 3$	1M
gemini-3-pro	Gemini	Google	1486	2$ / 12$	1M
gpt-5-mini	GPT	OpenAI	1390	0.25$ / 2$	400k
gpt-5.1	GPT	OpenAI	1437	1.25$ / 10$	400k

[2] As of 28^{st} of February 2026, data source: https://lmarena.ai/leaderboard/text.
[3] Cost input / output per 1 million tokens.

[4] https://developers.openai.com/api/docs/models/gpt-5-mini
[5] https://platform.claude.com/docs/en/about-claude/models/choosing-a-model

4.4 Experiment Results

In this section, we present our results related to the introductory hypotheses – the extensive experimental results can be found in the appendix. Apart from the main results, we also report intriguing findings encountered during the experiments. Initially, we analyze how the models themselves perform (**H1**): open-source versus closed-source (H1-1) and, within the closed-source models, small versus large (H1-2). Figure 2 shows that the paid models, both small and large, perform significantly better than the open-source models:

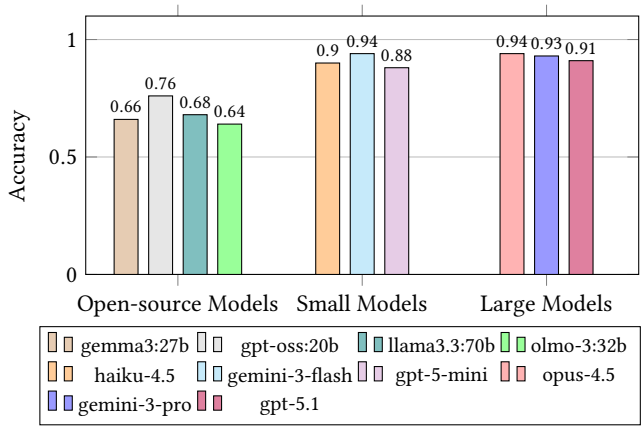

Figure 2: Overall model performance across all dataset sizes and error rates.

OpenAI's gpt-oss:20b performed best among the open-source models, being able to process larger dataset sizes and higher error rates with reasonable accuracy. For datasets above 10,000 observations, the open-source models were unable to generate valid responses within an acceptable time frame; they also started producing errors and hallucinated. For closed-source models, this threshold was much higher: for dataset sizes above 10,000 and even 40,000, they produced satisfactory results. Dataset sizes of 100,000 were mostly impossible to process because the maximum input tokens were reached, or errors and hallucinations began to occur frequently. As stated by LLM providers (see Section 4.3), smaller models should perform better in such clearly structured tasks than their larger counterparts. We observe a different outcome: the average accuracy of the smaller models is 93.82% in 56.44 s (Level 1) and 83.46% in 56.02 s (Level 2), which is slightly worse than that of the larger models, which achieve 96.95% in 49.63 s (Level 1) and 84.57% in 28.55 s (Level 2). **H2** claims that the accuracy will decrease when increasing the attribute cardinality represented by colors (5), car brands (50), and capital cities (100). Figure 3 shows this trend while holding dataset size (10,000) and error rate (1%) constant. For the colors dataset, all models achieve nearly 100% accuracy; for car brands and capital cities, accuracy declines to 95%. We also report the average accuracy across all models, error rates, and dataset sizes: 94.43% for colors, 93.44% for car brands, and 85.22% for capital cities. **H3** and **H4** are both tested on Level 1 and 2 datasets, applying the same logic: What happens to accuracy and execution time when increasing dataset sizes (H3) and error rates (H4)? Figure 4 holds the error rate at 1% and displays the accuracy of the models across the dataset sizes (1,000, 10,000, 20,000) for the Level 2 customer dataset. The closed-sourced models exhibit an obvious decline in accuracy. The general poor performance of the open-source models on Level 2 necessitates cautious interpretation of their results. Overall, the average accuracy for Level 2 is 90.00% (1,000), 87.89% (10,000), and 80.65% (20,000). Within the Level 1 tests, the results are 99.96% (100), 93.16% (1,000), and 75.00% (10,000). The respective average execution times across all models are 15.61 s (100), 57.34 s (1,000), and 115.52 s (10,000). The same procedure was applied to test the influence of an increasing error rate, with the dataset size fixed at 1,000 for the customer dataset (see Figure 5). The average accuracy at the smallest error rate (1 error in 1,000) is 100% for all models. When the error rate increases to 1% or even 10%, the results worsen – 91.35%, 82.22%, and 76.06% for the respective error rates of 0.1%, 1%, and 10% across all models and dataset sizes. Execution times increase with the error rates: 30.84 s, 34.20 s, and 64.71 s. To finalize this section, we highlight some general results that are not directly tied to the hypotheses. **Anthropic's Claude models delivered** – haiku-4.5 replied incredibly fast with some trade-offs in accuracy. Opus-4.5 was quick but with great accuracy even across large datasets. When analyzing the errors in more detail, we specifically tried to look out for patterns. Regarding the Level 1 dataset, the errors were random, but for Level 2, we observed that 70-90% of all not-found errors could be traced back to the street → city relation. This suggests that LLMs lack this level of depth of knowledge to accurately incorporate every street within each city worldwide. The city → country relation did not exhibit such problems.

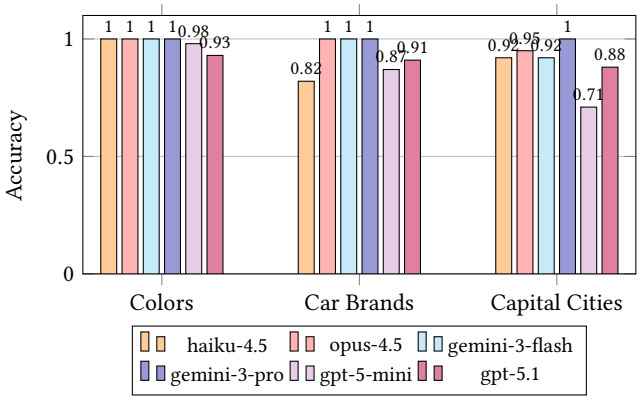

Figure 3: Level 1 Data - Dataset Size = 10,000, Error Rate = 1%, displayed are proprietary models only (small vs. large)

5 DISCUSSION

The experimental results we obtained suggest that LLMs can be used as a data error detection engine. However, during our investigation we faced **critical limitations** that, if not properly taken care of, can lead to undesired consequences. **Hallucination**, still an inherent problem for LLMs, can be observed in our tests, where some nonexistent words were reported as errors. Another important aspect is the trade-off between **scalability and cost**. The token-based pricing model applied by all major LLM providers does not favor large or high-frequency data inputs. Hence, it often makes

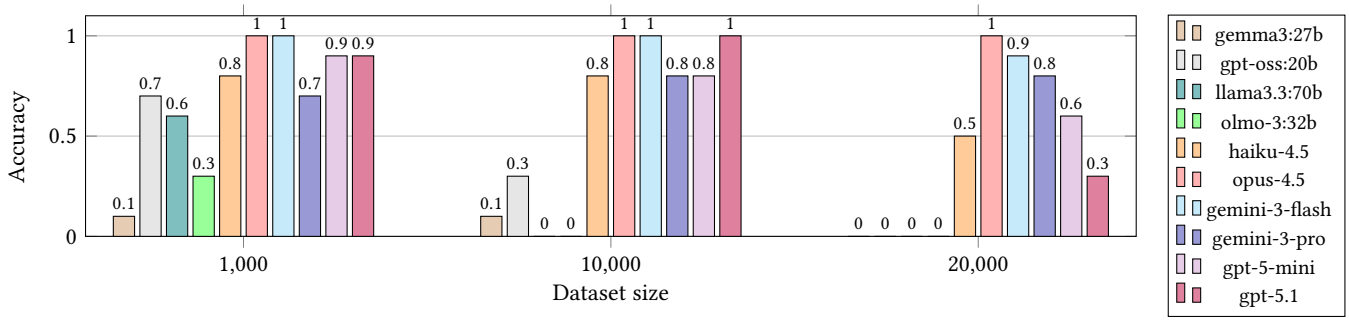

Figure 4: Customer Data - Error Rate = 1%

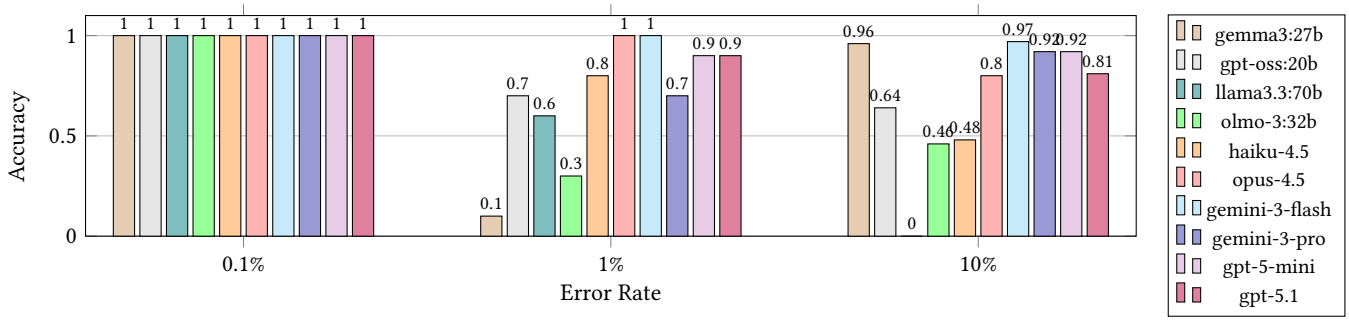

Figure 5: Customer Data - Dataset Size = 1,000

sense to start with open-source models first – in many scenarios, open models perform well at only a fraction of the cost. If these do not yield satisfactory results, the smaller, closed-source models should be tried next, as they are cheaper by 70-80% compared to the large models (see Table 1). Lastly, the three leading LLM providers, Anthropic, OpenAI, and Google's Gemini, pose a risk of **vendor lock-in**. Users may be subject to price changes, service outages, and abrupt model updates or retirements. Furthermore, passing confidential data via the APIs might be another point of concern, as it raises possible problems of data privacy, security, and compliance.

6 CONCLUSION AND OUTLOOK

Despite the challenges, the advantages of applying LLMs for data error detection are obvious. Our experiments provide strong evidence for our initial hypotheses: we confirmed hypothesis **H1-1**, which states that closed-source models outperform open-source models. Within the closed-source models, we showed that the larger models surprisingly excelled over smaller ones, thereby refuting hypothesis **H1-2**. Additionally, we proved hypothesis **H2**, inferring a negative correlation between attribute cardinality and accuracy. The hypotheses **H3** and **H4** were also confirmed, showing that both increasing dataset size and error rate lead to a decrease in accuracy and an increase in execution time. LLMs exhibit remarkable accuracy and acceptable execution times; however, significant limitations must be addressed: hallucinations, trade-offs between scalability and cost, and reliance on a limited number of providers. To improve these systems, future research should focus on implementing mitigation strategies: a **human-in-the-loop validation** process helps counter hallucination and achieve higher trustworthiness. Furthermore, adding **post-processing verification** methods can make the results more reliable. We recommend initially prioritizing open-source models by implementing a **looped batch-processing mechanism** before considering smaller, closed-source models. We encourage LLM providers to investigate new pricing models tailored for data-intensive tasks. Such adjustments would promote broader experimentation with LLMs in the data domain and accelerate the integration of these models into more data pipelines.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proc. VLDB Endow.* 9, 12 (2016), 993–1004.

[2] Fatemeh Ahmadi, Julian Paulußen, and Ziawasch Abedjan. 2025. Demonstrating Matelda for Multi-Table Error Detection. *Proc. VLDB Endow.* 18, 12 (2025), 5379–5382.

[3] Maria Chernigovskaya, Abdulrahman Nahhas, André Hardt, and Klaus Turowski. 2025. Challenges and Limitations of Leveraging LLMs for Anomaly Detection: Insights from a Manufacturing Case Study. In *International Conference on Smart Business Technologies*. Springer Nature Switzerland, 250–264.

[4] Wei-Lin Chiang, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. *Forty-first International Conference on Machine Learning* (2024).

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1*. Association for Computational Linguistics, 4171–4186.

[6] Anna-Christina Glock, Christine Dominka-Kiss, Philipp Korom, and Lisa Ehrlinger. 2025. Detecting and Cleaning Errors in Personal Contact Information with Large Language Models.. In *Proceedings of the VLDB Endowment. ISSN 2150-8097*. VLDB Endowment., 8097.

[7] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. 2003. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Semant.* 1, 1 (2003), 7–26.

[8] Johann Schopplich. 2025. *TOON (Version 2.1.0).* https://github.com/toon-format/toon GitHub repository.

[9] Aodong Li, Yunhan Zhao, Chen Qiu, Marius Kloft, Padhraic Smyth, Maja Rudolph, and Stephan Mandt. 2024. Anomaly Detection of Tabular Data Using LLMs. *CoRR* abs/2406.16308 (2024). arXiv:2406.16308

[10] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

[11] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 3111–3119.

[12] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 1532–1543.

[13] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45.

[14] Jing Ren, Tao Tang, Hong Jia, Ziqi Xu, Haytham Fayek, Xiaodong Li, Suyu Ma, Xiwei Xu, and Feng Xia. 2025. Foundation Models for Anomaly Detection: Vision and Challenges. *AI Mag.* 46, 4 (2025).

[15] Gerard Salton and Chris Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manag.* 24, 5 (1988), 513–523.

[16] Richard Y. Wang and Diane M. Strong. 1996. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manag. Inf. Syst.* 12, 4 (1996), 5–33.

[17] Tiankai Yang, Yi Nian, Li Li, Ruiyao Xu, Yuangang Li, Jiaqi Li, Zhuo Xiao, Xiyang Hu, Ryan A. Rossi, Kaize Ding, Xia Hu, and Yue Zhao. 2025. AD-LLM: Benchmarking Large Language Models for Anomaly Detection. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025 (Findings of ACL)*. Association for Computational Linguistics, 1524–1547.

A APPENDIX

Table 2: Extensive Results for Level 1 - Colors

Model	Error Rate	Dataset Size					
		100		1.000		10.000	
		Acc.	Time	Acc.	Time	Acc.	Time
gemma3:27b	1%	1	2,2	1	4,8	timeout	
	10%	1	2,53	0,75	4,97	timeout	
	25%	1	3,01	timeout		timeout	
gpt-oss:20b	1%	1	3,66	0,9	3,71	0,47	41,77
	10%	1	4,91	0,9	8,68	0,039	77,4
	25%	1	8,8	0,932	42,83	error	
llama3.3:70b	1%	1	21,1	1	25,82	0,64	198,59
	10%	1	17,5	0,95	132,56	0,23	467,9
	25%	1	54,24	0,91	322,28	timeout	
olmo-3:32b	1%	1	13,7	0,7	257,26	timeout	
	10%	1	39,56	0,81	191,12	timeout	
	25%	1	126,69	timeout		timeout	
haiku-4.5	1%	1	0,66	1	2,15	1	5,8
	10%	1	1,73	1	2,85	0,997	13,87
	25%	1	1,78	0,968	8,39	error	
opus-4.5	1%	1	2,52	1	2,71	1	8,77
	10%	1	2,59	1	4,7	0,995	54,48
	25%	1	2,98	1	8,7	0,998	149,05
gemini-3-flash	1%	1	5,05	1	7,72	1	94,64
	10%	1	6,56	1	96,02	1	79,56
	25%	1	22,38	1	161,59	0,9996	328,78
gemini-3-pro	1%	1	8,15	1	8,15	1	42,2
	10%	1	20,23	1	20,23	1	274,95
	25%	1	25,46	1	25,46	1	433,2
gpt-5-mini	1%	1	2,7	1	10,13	0,98	80,54
	10%	1	10,93	1	30,73	0,57	46,43
	25%	1	16,94	1	126,72	error	
gpt-5.1	1%	1	2,12	1	1,79	0,93	6,55
	10%	1	1,75	1	4,01	0,995	25,44
	25%	1	2,94	1	7,19	error	

Table 3: Extensive Results for Level 1 - Car Brands

Model	Error Rate	Dataset Size					
		100		1.000		10.000	
		Acc.	Time	Acc.	Time	Acc.	Time
gemma3:27b	1%	1	2,19	0,8	3,06	error	
	10%	1	2,54	0,75	5,07	error	
	25%	0,96	2,91	0,47	6,37	error	
gpt-oss:20b	1%	1	2,68	1	38,74	0,37	62,66
	10%	1	5,45	0,94	34,65	error	
	25%	1	5,68	0,98	31,45	error	
llama3.3:70b	1%	1	21,93	1	56,59	timeout	
	10%	1	17,66	0,83	122,83	timeout	
	25%	1	35,83	0,69	240,03	timeout	
olmo-3:32b	1%	1	35,87	0,9	492,07	timeout	
	10%	1	79,23	timeout		timeout	
	25%	1	120,83	timeout		timeout	
haiku-4.5	1%	1	0,74	1	1	0,82	3,92
	10%	1	0,91	1	3,26	0,54	20,54
	25%	1	1,87	0,94	6,49	error	
opus-4.5	1%	1	1,81	1	2,98	1	8,79
	10%	1	2,72	1	5,23	0,88	45,44
	25%	1	3,05	1	9,26	0,99	174,22
gemini-3-flash	1%	1	5,96	1	65,88	1	97,03
	10%	1	6,61	1	76,16	0,99	151,17
	25%	1	51,04	1	163,99	0,85	183,92
gemini-3-pro	1%	1	9,53	1	11,05	1	35,47
	10%	1	14,49	1	55,42	1	235,55
	25%	1	16,82	1	145,44	0,997	491,28
gpt-5-mini	1%	1	5,22	1	22,3	0,87	90,65
	10%	1	20,2	1	103,69	0,169	110,95
	25%	1	29,75	1	100,04	error	
gpt-5.1	1%	1	1,16	1	2,07	0,91	5,76
	10%	1	1,76	1	3,6	0,76	24,07
	25%	1	1,61	1	9,03	0,66	57,09

Table 4: Extensive Results for Level 1 - Capital Cities

Model	Error Rate	Dataset Size					
		100		1.000		10.000	
		Acc.	Time	Acc.	Time	Acc.	Time
gemma3:27b	1%	1	2,22	1	3,25	0,15	11,84
	10%	1	2,61	0,61	4,72	0,01	25,37
	25%	1	3,08	0,22	4,58	error	
gpt-oss:20b	1%	1	5,08	0,9	3,61	0,35	47,33
	10%	1	7,87	0,79	53,47	0,01	22,27
	25%	1	14,79	0,78	46,41	error	
llama3.3:70b	1%	1	16,68	0,9	59,4	0,13	36,17
	10%	1	18,13	0,63	100,11	0,07	316,39
	25%	1	36,97	0,55	207,82	timeout	
olmo-3:32b	1%	1	36,7	1	123,57	timeout	
	10%	1	46,46	timeout		timeout	
	25%	1	59,27	timeout		timeout	
haiku-4.5	1%	1	0,78	1	1,42	0,92	8,58
	10%	1	1,62	0,9	3,37	0,54	16,39
	25%	1	1,21	0,86	8,51	error	
opus-4.5	1%	1	1,85	1	2,56	0,95	8,09
	10%	1	2,64	1	5,42	0,84	52,58
	25%	1	3,22	1	8,9	0,41	47,48
gemini-3-flash	1%	1	3,91	1	8,06	0,92	168,69
	10%	1	6,8	1	94,31	0,99	330,26
	25%	1	14,24	1	180,16	0,46	365,07
gemini-3-pro	1%	1	24,67	1	79,89	1	68,43
	10%	1	69,35	1	61,37	0,999	334,04
	25%	1	29,11	1	156,26	0,997	360,11
gpt-5-mini	1%	1	11,39	1	19,69	0,71	48,69
	10%	1	18,74	1	66,95	error	
	25%	1	17,45	1	156,42	error	
gpt-5.1	1%	1	1,14	1	2,83	0,88	4,93
	10%	1	1,61	1	5,08	0,59	21,44
	25%	1	1,72	1	7,57	0,75	76,16

Table 5: Extensive Results for Level 2 - Customer Data

Model	Error Rate	Dataset Size					
		1.000		10.000		20.000	
		Acc.	Time	Acc.	Time	Acc.	Time
gemma3:27b	0, 1%	1	4,54	error		timeout	
	1%	0,1	4,49	0,1	18,55	timeout	
	10%	0,96	44,6	0,19	334,39	timeout	
gpt-oss:20b	0, 1%	1	19,16	1	78,91	timeout	
	1%	0,7	31,92	0,3	67,87	timeout	
	10%	0,64	27,51	0,03	69,99	timeout	
llama3.3:70b	0, 1%	1	166,95	timeout		timeout	
	1%	0,6	261,58	timeout		timeout	
	10%	timeout		timeout		timeout	
olmo-3:32b	0, 1%	1	241,09	0	252,76	timeout	
	1%	0,3	119,76	0	291,96	timeout	
	10%	0,46	215,3	0,01	151,08	timeout	
haiku-4.5	0, 1%	1	4,06	1	5,86	1	10,02
	1%	0,8	4,56	0,8	4,98	0,5	8,43
	10%	0,48	7,42	0,73	7,38	error	
opus-4.5	0, 1%	1	10,1	1	10,27	1	7,23
	1%	1	10,86	1	9	1	14,36
	10%	0,8	14,04	0,68	18,6	0,62	17,29
gemini-3-flash	0, 1%	1	28,49	1	77,51	1	144,6
	1%	1	11,81	1	76,31	0,9	120,45
	10%	0,97	116,64	0,85	41,89	0,83	63,86
gemini-3-pro	0, 1%	1	53,09	1	29,7	1	44,74
	1%	0,7	60,86	0,8	28,84	0,8	36,72
	10%	0,92	76,24	0,78	71,15	0,76	133,9
gpt-5-mini	0, 1%	1	30,39	1	32,98	1	31,51
	1%	0,9	72,49	0,8	43,86	0,6	81,12
	10%	0,92	113	0,63	98,1	0,65	255,51
gpt-5.1	0, 1%	1	5,78	1	4,57	1	9,6
	1%	0,9	4,69	1	12,84	0,3	13,5
	10%	0,81	21,59	0,75	11,32	0,75	32,1