Automated Classification of REST API Test Failures: A Comparison of LLM and non-LLM Classifiers

Selen Öznur[1,*], Tolgahan Bardakci[1,2] and Serge Demeyer[1,2]

[1]University of Antwerp, Belgium
[2]Flanders Make, Belgium

Abstract

LLM-based agents are increasingly used to generate and amplify REST API tests, but failures produced during this process still require manual inspection to determine their origin, that is, whether they arise from the generated test or from the API under test. We investigate the effectiveness of LLM and non-LLM classifiers for classifying the origin of these failures. We evaluate three LLMs under four prompting strategies on 1,725 manually labeled failed tests from five REST APIs and compare them with two non-LLM classifiers, Naive Bayes and a Decision Tree. Our results show that a depth-8 Decision Tree achieves the best Macro-F1 (0.523), outperforming the highest-scoring LLM configuration, Claude Haiku 4.5 with few-shot prompting (0.399). In our setting, these results show that a non-LLM classifier trained on labeled failures can be competitive with LLM classifiers for test failure classification.

Keywords

REST APIs, Software Testing, Failure Classification, Large Language Models, Prompt Engineering

1. Introduction

LLM-based agents are increasingly used to generate and amplify REST API test suites [1, 2, 3], but some generated tests fail during execution. These failures may indicate real defects in the API under test or originate from the generated test itself, such as a missing precondition, invalid input, or incorrect assertion. Identifying the root cause requires inspecting the failing test, its error message, and the relevant API specification.

Besjes et al. [2] propose five failure categories for these root causes: Semantically Incorrect, Wrong Assertion, Missing Information, Wrong Input, and Bug. As generated test suites grow, manually applying this taxonomy requires substantial effort [4], and it becomes difficult to scale.

LLMs are a natural candidate for automating this classification because they can use the failing test, error message, and API specification to assign a failure category. However, it is unclear whether an LLM is needed. The inputs are text, and non-LLM classifiers work well on similar text classification problems [5]. Therefore, in this study, we present LLM and non-LLM classifiers for automated failure classification. We investigate two questions:

- RQ1: *How effective are LLM and non-LLM classifiers at classifying failures in LLM-generated REST API tests?*
- RQ2: *How does performance vary across failure categories?*

We evaluate GPT-4o-mini [6], GPT-4o [7], and Claude Haiku 4.5 [8] under four prompting strategies. We compare the LLM classifiers with two non-LLM classifiers, Naive Bayes and a Decision Tree, which use term frequency–inverse document frequency (TF-IDF) representations of the same input data. We use a dataset that contains 1,725 manually labeled failed tests from five REST APIs [9]. We also show how performance varies across failure categories.

BENEVOL2026: Belgium-Netherlands Software Evolution Workshop 26–27 November, 2026, Madrid, Spain
*Corresponding author.
35selenoznur@gmail.com (S. Öznur); tolgahan.bardakci@uantwerpen.be (T. Bardakci); serge.demeyer@uantwerpen.be (S. Demeyer)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Related Work

REST API Testing and Amplification. REST API testing has been widely studied [10]. RESTler [11], EvoMaster [12], and RESTest [13] present various approaches for automated REST API testing. Nooyens et al. [1] used single- and multi-agent LLM systems to amplify REST API test suites. Besjes et al. [2] extend this comparison across five applications. Researchers have also proposed coverage criteria and supporting tools for RESTful APIs [14, 15].

LLM-Based Failure and Bug Classification. Beyond REST API testing, researchers have studied failure classification broadly in software testing. Akli et al. [16] and Lin et al. [17] classify flaky tests into root-cause categories. Rahman et al. [18] show that evaluation design can substantially affect reported classifier performance when classifying flaky test categories. Du et al. [19] and Koyuncu [20] study prompt-based bug-report classification.

LLM and non-LLM Classifiers. Prompt choice can affect LLM performance [21]. Hannan et al. [22] show that few-shot example selection influences vulnerability detection. Tang et al. [23] report that additional few-shot examples can even reduce performance. Non-LLM classifiers such as Naive Bayes and the Decision Tree algorithm can remain competitive on software engineering tasks [24]. Gao et al. [5] report that simple non-LLM classifiers can perform competitively when predicting test-failure causes from execution logs.

Research gap. Prior work uses LLMs to generate and amplify REST API tests [1, 2, 3], classifies flaky tests and bug reports [16, 17, 18, 19, 20], and shows that simple classifiers can compete with complex ones on test-failure logs [5]. To our knowledge, however, no study has compared LLM and non-LLM classifiers on the failures of LLM-generated REST API tests, nor evaluated them on endpoints unseen during training. We address this gap.

3. Method

3.1. Dataset

We use the dataset presented by Besjes et al. [2], containing 1,735 failed test cases from Spotify [25], Google Drive [26], PetStore [27], VAmPI [28], and Restful-Booker [29]. Each test case in the dataset is labeled as Semantically Incorrect, Wrong Assertion, Missing Information, Wrong Input, or Bug.

Before classification, we checked the reconstructed dataset and ground-truth labels. We corrected 223 records where the source file did not match the HTTP request in the execution log. We also found 211 authentication failures labeled inconsistently across the two agent configurations. They were labeled as Missing Information in one configuration and as Semantically Incorrect or Wrong Assertion in the other. Since these failures resulted from an invalid or expired shared token rather than issues with the tests themselves, we relabeled all 211 records as Missing Information. After excluding the 10 few-shot examples described in Section 3.3, the final dataset contained 1,725 records.

3.2. Classifier Inputs

Each classifier receives the test code, endpoint OpenAPI specification, execution error message, and HTTP method. We exclude the human explanation to prevent label leakage, and the system and agent configuration to avoid dataset-specific shortcuts.

3.3. LLM Classifiers

We use GPT-4o-mini [6], GPT-4o [7], and Claude Haiku 4.5 [8] with four prompting strategies: zero-shot, few-shot, definition-guided, and definition-guided-v2.

Zero-shot explains the task and lists the five category names. *Few-shot* uses the same prompt and adds 10 labeled examples, with two examples for each category. *Definition-guided* gives a definition for each category instead of examples. *Definition-guided-v2* uses revised definitions for all five categories. The biggest changes are for Semantically Incorrect and Wrong Assertion, because the audit showed that they were frequently confused. The prompt, *definition-guided-v2*, specifies Semantically Incorrect as a failure to establish the scenario, and Wrong Assertion as an incorrect expected outcome once the scenario is established. All experiments use temperature 0 to make the predictions reproducible and to avoid randomness when comparing models and prompting strategies.

While the five categories come from Besjes et al. [2], the prompt texts are our own. The LLMs are not trained, so we run each of the twelve configurations once on all 1,725 evaluation records (excluding the 10 few-shot examples).

3.4. Non-LLM Classifiers

Prior work has evaluated Naive Bayes and Decision Trees for software issue classification [30], while a recent mapping study identifies these classifiers and TF-IDF as commonly used techniques in this area [31]. Accordingly, we use Naive Bayes and a Decision Tree as our non-LLM classifiers. We represent the input with TF-IDF using unigrams and bigrams to capture words and phrases. We set the minimum document frequency to 2 and limit the vocabulary to 20,000 features. The Decision Tree uses balanced class weights and a maximum depth of 8, fixed a priori to limit complexity rather than tuned on held-out performance. A shallow tree also keeps its rules readable. Naive Bayes uses smoothing with $\alpha = 1$ and no class weighting. We likewise set the TF-IDF settings and the smoothing value a priori, without tuning. We use a fixed seed (42) for cross-validation splitting. Since the non-LLM classifiers are trained on labeled examples, we use 5-fold cross-validation so each record is predicted without its own label during training.

The dataset covers 142 API endpoint groups, with each group containing all tests targeting the same endpoint across agent configurations. To prevent overlap between training and test data, we keep records from the same endpoint in a single fold. This allows us to evaluate the classifiers on endpoints they have not seen during training. We use endpoint-grouped 5-fold cross-validation with seed 42 for the primary comparison with the LLM classifiers. We evaluate each non-LLM classifier once, and every record is predicted by a model trained on other endpoints. We also performed standard record-level stratified 5-fold cross-validation for comparison, where records from the same endpoint may appear in both training and test folds (Section 4.1).

3.5. Evaluation Metrics

Since the failure categories are imbalanced, as described in Section 3.1, we use Macro-F1 as our primary metric. It weights each category equally, preventing larger categories from dominating the score.

We report precision, recall, accuracy, and Cohen's κ as complementary metrics. Cohen's κ measures agreement while accounting for agreement expected by chance. We also present all metrics by category for the highest-scoring classifiers.

4. Results

4.1. RQ1: How effective are LLM and non-LLM classifiers at classifying failures in LLM-generated REST API tests?

Table 1 reports all classifier scores. The highest-scoring classifiers and the corresponding methods are shown in bold.

As shown in Table 1, the Decision Tree achieves the highest Macro-F1 score (0.523). In comparison, Claude Haiku 4.5 with few-shot prompting, the highest-scoring LLM configuration, achieves a Macro-F1 score of 0.399. Naive Bayes reaches 0.310, so the advantage holds for our Decision Tree configuration

Table 1
Performance of all evaluated classifiers, grouped by classifier. Best results in bold.

Model	Prompt	Macro-F1	Acc.	Precision	Recall	κ
Decision Tree	—	**0.523**	**0.586**	**0.541**	**0.520**	**0.456**
Naive Bayes	—	0.310	0.437	0.321	0.336	0.223
GPT-4o-mini	zero-shot	0.280	0.334	0.296	0.324	0.166
	few-shot	0.236	0.269	0.327	0.318	0.120
	definition-guided	0.174	0.211	0.322	0.251	0.066
	definition-guided-v2	0.279	0.330	0.306	0.302	0.149
GPT-4o	zero-shot	0.300	0.361	0.403	0.335	0.183
	few-shot	0.385	0.426	0.395	0.391	0.240
	definition-guided	0.337	0.390	0.376	0.350	0.215
	definition-guided-v2	0.332	0.401	0.344	0.346	0.202
Claude Haiku 4.5	zero-shot	0.275	0.340	0.368	0.314	0.159
	few-shot	**0.399**	**0.448**	**0.427**	**0.428**	**0.252**
	definition-guided	0.320	0.372	0.427	0.345	0.193
	definition-guided-v2	0.283	0.373	0.381	0.306	0.156

Table 2
Per-category performance of the two highest-scoring classifiers. Best value per category and metric in bold.

	Decision Tree			Claude Haiku 4.5 + few-shot		
Category	Precision	Recall	F1	Precision	Recall	F1
Missing Information	**0.77**	0.70	**0.73**	0.47	**0.81**	0.59
Semantically Incorrect	**0.61**	**0.60**	**0.60**	0.48	0.24	0.32
Wrong Input	**0.66**	**0.51**	**0.58**	0.42	0.23	0.30
Wrong Assertion	0.34	**0.53**	**0.42**	**0.41**	0.29	0.34
Bug	0.32	0.26	0.29	**0.36**	**0.57**	**0.44**

but not for Naive Bayes. These results apply to the configurations and evaluation procedure studied here; they do not show that Decision Trees are inherently better than LLMs.

Evaluation strategy also affects the non-LLM classifiers. Under record-level cross-validation, the Decision Tree and Naive Bayes achieve Macro-F1 scores of 0.689 and 0.474, compared with 0.523 and 0.310 under endpoint-grouped cross-validation. This suggests that record-level splitting can overestimate performance when related records from the same endpoint appear across folds, while endpoint-grouped evaluation provides a more conservative estimate of generalization to unseen endpoints.

4.2. RQ2: How does performance vary across failure categories?

Table 2 reports the metrics organized by the failure category. We present the highest-scoring classifiers: Decision Tree and Claude Haiku 4.5 with few-shot prompting.

Per-category performance differs between the Decision Tree and the highest-scoring LLM configuration. The Decision Tree achieves higher F1 scores for four categories, while Claude Haiku 4.5 with few-shot prompting performs better for the Bug category.

Bug is the rarest category (5.2% of records, 90 of 1,725), but it signals a defect in the API under test rather than in the generated test. Here Claude Haiku 4.5 with few-shot prompting finds more than twice as many bugs as the Decision Tree (recall 0.57 vs. 0.26), although precision is low for both (0.36 vs. 0.32). If the goal is to miss as few real bugs as possible, this recall may matter more than the overall Macro-F1; if the goal is the best overall labeling, the Decision Tree is preferable. This suggests combining the two (Section 7).

We present the full table in our reproduction package [32] under the "Full Table" directory.

5. Data Availability

The reproduction package is available on Zenodo as an artifact [32]. It contains the dataset's raw and audited versions, the classification scripts with the prompts used, and the scripts and images produced during dataset analysis. It also includes the full per-API and per-category results for every classification configuration, which Tables 1 and 2 summarize. Finally, it provides the rule-based procedure used to audit the ground truth.

6. Limitations

Our results should be read with four limitations in mind.

First, we train the non-LLM classifiers via cross-validation, while we use the LLMs without training on the dataset. Both setups hide each record's label during prediction, but are not equivalent. Our conclusions therefore apply to the configurations and evaluation procedure studied here, not to LLMs or Decision Trees in general.

Second, the dataset covers only five REST APIs and one failure taxonomy, so our findings may not generalize beyond them.

Third, the 10 few-shot examples share similarities with 112 of 1,725 evaluation records (6.5%). Thus, few-shot LLMs may access endpoint-specific information that the other classifiers cannot, potentially favoring the few-shot results.

Finally, we tested three LLMs from two providers; newer models (e.g., GPT-5) and open-weight models (e.g., Llama, Qwen) were not evaluated and may behave differently.

7. Conclusion

We studied the automated classification of failed API test cases using LLM and non-LLM classifiers. The Decision Tree achieved the highest Macro-F1 score (0.523), outperforming the highest-scoring LLM configuration, Claude Haiku 4.5 with few-shot prompting (0.399). The advantage does not extend to both non-LLM classifiers, since Naive Bayes reached only 0.310. These findings are early evidence that LLMs are not a silver bullet for this task and that a simple classifier deserves to be the first baseline.

The approaches also differ in cost. Non-LLM classifiers can be trained quickly on a laptop without any per-prediction fees, while each LLM configuration required 1,725 API calls (20,700 across all twelve configurations). However, the LLM identified more real bugs (Section 4.2), suggesting that the two approaches could be used together. Future work could explore combining them by using non-LLM classifiers for straightforward cases and referring uncertain cases to an LLM or a human. We also plan to evaluate additional APIs, newer and open-weight LLMs, and fine-tuned models. Finally, adding an explicit authentication-status feature could help distinguish authentication failures from the underlying failure categories.

Declaration on Generative AI

The authors used Claude (Anthropic) and ChatGPT (OpenAI) to draft, revise, and format the manuscript and references. We reviewed and edited the final text and take full responsibility for publication.

Acknowledgments

This work is supported by the Research Foundation Flanders (FWO) via the BaseCamp Zero Project under Grant number S000323N.

References

[1] R. Nooyens, T. Bardakci, M. Beyazıt, S. Demeyer, Test Amplification for REST APIs via Single and Multi-agent LLM Systems, in: S. Bonfanti, G. A. Papadopoulos (Eds.), Testing Software and Systems, volume 16107, Springer Nature Switzerland, Cham, 2026, pp. 161–177. doi:10.1007/978-3-032-05188-2_11.

[2] J. Besjes, R. Nooyens, T. Bardakci, M. Beyazit, S. Demeyer, Agentic LLMs for REST API Test Amplification: A Comparative Study Across Cloud Applications, 2025. doi:10.48550/arXiv.2510.27417. arXiv:2510.27417.

[3] T. Stennett, M. Kim, S. Sinha, A. Orso, AutoRestTest: A Tool for Automated REST API Testing Using LLMs and MARL, in: 2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, Ottawa, ON, Canada, 2025, pp. 21–24. doi:10.1109/ICSE-Companion66252.2025.00015.

[4] C. Jordan, P. Foth, M. Fruth, A. Pretschner, Why Did the Test Execution Fail? Failure Classification Using Association Rules (Practical Experience Report), in: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), IEEE, Charlotte, NC, USA, 2022, pp. 517–528. doi:10.1109/ISSRE55969.2022.00056.

[5] Z. Gao, Z. Xue, X. Hu, W. Shang, X. Xia, Easy over Hard: A Simple Baseline for Test Failures Causes Prediction, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 306–317. doi:10.1145/3663529.3663850. arXiv:2405.02922.

[6] OpenAI, GPT-4o mini, https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

[7] OpenAI, GPT-4o, https://openai.com/index/hello-gpt-4o/, 2024.

[8] Anthropic, Claude Haiku 4.5, https://www.anthropic.com/claude/haiku, 2025.

[9] J. Besjes, R. Nooyens, T. Bardakci, M. Beyazit, S. Demeyer, Dataset, https://zenodo.org/records/19251015, 2025. doi:10.5281/zenodo.19251015.

[10] A. Golmohammadi, M. Zhang, A. Arcuri, Testing RESTful APIs: A Survey, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–41. doi:10.1145/3617175.

[11] V. Atlidakis, P. Godefroid, M. Polishchuk, RESTler: Stateful REST API Fuzzing, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 748–758. doi:10.1109/ICSE.2019.00083.

[12] A. Arcuri, EvoMaster: Evolutionary Multi-context Automated System Test Generation, in: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), 2018, pp. 394–397. doi:10.1109/ICST.2018.00046. arXiv:1901.04472.

[13] A. Martin-Lopez, S. Segura, A. Ruiz-Cortés, RESTest: Automated black-box testing of RESTful web APIs, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, Virtual Denmark, 2021, pp. 682–685. doi:10.1145/3460319.3469082.

[14] A. Martin-Lopez, S. Segura, A. Ruiz-Cortés, Test coverage criteria for RESTful web APIs, in: Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, ACM, Tallinn Estonia, 2019, pp. 15–21. doi:10.1145/3340433.3342822.

[15] D. Corradini, A. Zampieri, M. Pasqua, M. Ceccato, Restats: A Test Coverage Tool for RESTful APIs, in: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, Luxembourg, 2021, pp. 594–598. doi:10.1109/ICSME52107.2021.00063.

[16] A. Akli, G. Haben, S. Habchi, M. Papadakis, Y. L. Traon, Predicting Flaky Tests Categories using Few-Shot Learning, 2022. doi:10.48550/arXiv.2208.14799. arXiv:2208.14799.

[17] S. Lin, R. Z. H. Liu, L. Tahvildari, FlaKat: A Machine Learning-Based Categorization Framework for Flaky Tests, 2024. doi:10.48550/arXiv.2403.01003. arXiv:2403.01003.

[18] S. Rahman, S. Dutta, A. Shi, Understanding and Improving Flaky Test Classification, Proceedings of the ACM on Programming Languages 9 (2025) 1345–1371. doi:10.1145/3763098.

[19] X. Du, Z. Liu, C. Li, X. Ma, Y. Li, X. Wang, LLM-BRC: A large language model-based bug report classification framework, Software Quality Journal 32 (2024) 985–1005. doi:10.1007/

s11219-024-09675-3.
[20] A. Koyuncu, Exploring Fine-Grained Bug Report Categorization with Large Language Models and Prompt Engineering: An Empirical Study, ACM Transactions on Software Engineering and Methodology 35 (2026) 1–47. doi:10.1145/3736408.
[21] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. Kroiz, F. Li, H. Tao, A. Srivastava, H. D. Costa, S. Gupta, M. L. Rogers, I. Goncearenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle, P. Resnik, The Prompt Report: A Systematic Survey of Prompt Engineering Techniques, 2025. doi:10.48550/arXiv.2406.06608. arXiv:2406.06608.
[22] M. A. Hannan, R. Ni, C. Zhang, L. Jia, R. Mangal, C. S. Pasareanu, On the Difficulty of Selecting Few-Shot Examples for Effective LLM-based Vulnerability Detection, in: Proceedings 2026 Workshop on LLM Assisted Security and Trust Exploration, 2026. doi:10.14722/last-x.2026.23025. arXiv:2510.27675.
[23] Y. Tang, D. Tuncel, C. Koerner, T. Runkler, The Few-shot Dilemma: Over-prompting Large Language Models, in: 2025 3rd International Conference on Foundation and Large Language Models (FLLM), 2025, pp. 134–141. doi:10.1109/FLLM67465.2025.11391015. arXiv:2509.13196.
[24] W. Cunha, L. Rocha, M. A. Gonçalves, A thorough benchmark of automatic text classification: From traditional approaches to large language models, 2025. doi:10.48550/arXiv.2504.01930. arXiv:2504.01930.
[25] Spotify, Spotify, https://developer.spotify.com/documentation/web-api, 2025.
[26] Google for Developers, Google Drive, https://developers.google.com/workspace/drive, 2025.
[27] S. Software, PetStore, https://petstore.swagger.io/, 2025.
[28] erev0s, VAmPI, https://github.com/erev0s/VAmPI, 2026.
[29] M. Winteringham, Restful-Booker, https://restful-booker.herokuapp.com/, 2025.
[30] N. Pandey, D. K. Sanyal, A. Hudait, A. Sen, Automated classification of software issue reports using machine learning techniques: An empirical study, Innovations in Systems and Software Engineering 13 (2017) 279–297. doi:10.1007/s11334-017-0294-1.
[31] M. Laiq, F. Dobslaw, Automatic techniques for issue report classification: A systematic mapping study, Automated Software Engineering 33 (2026) 72. doi:10.1007/s10515-026-00616-x.
[32] S. Öznur, T. Bardakci, S. Demeyer, Reproduction Package: "Automated Classification of REST API Test Failures: A Comparison of LLM and non-LLM Classifiers", 2026. doi:10.5281/ZENODO.22686884.