Accepted for the EASE 2025. This is the camera-ready author version of the paper, the final version will be accessible online.

Large Language Models for API Classification: An Explorative Study

Gabriel Morais*
gabrielglauber.morais@uqar.ca
Université du Québec à Rimouski (UQAR)
Lévis, Québec, Canada

Edwin Lemelin*
Université du Québec à Rimouski (UQAR)
Lévis, Québec, Canada
Université Laval
Québec, Québec, Canada

Mehdi Adda
mehdi_adda@uqar.ca
Université du Québec à Rimouski (UQAR)
Lévis, Québec, Canada

Dominik Bork
dominik.bork@tuwien.ac.at
TU Wien
Vienna, Austria
Université du Québec à Rimouski (UQAR)
Lévis, Québec, Canada

Abstract

Linking APIs to the business functions they implement is crucial for handling software operations, especially during recovery from disasters or outages. In this context, the speed and accuracy of operators in linking them impact response time during mission-critical operation activities. Besides, this linkage is essential to designing preventive actions, such as resilience strategies. Automatic API classification using Large Language Models (LLMs) may simplify and speed up APIs-business function linkage. However, previous studies unveiled the barriers practitioners face when deciding on and adopting LLMs in software engineering (SE) tasks due to a lack of guidance for non-experts. This paper aims to lower barriers to using LLMs by systems operators and site reliability engineers (SREs), focusing on the API classification task in the context of operational activities. Based on three cases from the finance industry, we extracted requirements for LLM usage, and assessed 14 recently released LLMs on this task. Our results demonstrate that LLMs accurately classify APIs using business function targets with an *F1–Score* of 89.5 for the leading LLM without requiring specific LLM expertise and resource-intensive fine-tuning. Besides, our findings on LLMs' performance and reliability mark a significant advancement in comparing open and closed-source and general and domain-specific LLMs in an SE classification task. Eventually, our experiments yield practical guidance for implementing LLMs in this context. Artifacts used in and generated by the experiments are publicly available at https://bit.ly/llms4apiclassification.

Keywords

Large Language Models, Software Engineering Operations, API Classification

ACM Reference Format:
Gabriel Morais, Edwin Lemelin, Mehdi Adda, and Dominik Bork. 2025. Large Language Models for API Classification: An Explorative Study. In *Proceedings of the 2025 Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3756681.3756997

1 Introduction

Application Programming Interfaces (APIs) are used in software engineering (SE) to define interactions between applications [31], serving as a contract between API providers and consumers [44]. The proliferation of APIs, fostered by adopting the Microservices Architecture (MSA) [22], complexifies systems' operation [51]. MSA-based systems are formed by arranging numerous APIs, requiring new governance approaches, including organization-wide systems knowledge in an environment where the system architecture evolves quickly [27]. In this context, operators and SREs may be clueless about which APIs are involved in which business function during operational activities. This fact is exacerbated when facing system disruptions, as impacts are considered from the business perspective. Thus, operators must link business functionalities to system parts to fix failures and support the design of recovery from disaster strategies [35]. Manually identifying APIs implementing business functions is costly; automatic methods and classification mechanisms may facilitate this process [28].

Large Language Models (LLMs) have been considered as change players in various SE tasks. Therefore, we hypothesize that applying pre-trained LLMs without additional fine-tuning is an effective solution for API classification. Still, understanding their impact on SE tasks and business mission-critical processes is in its early stages [10, 14]. Besides, the existing literature on LLMs primarily focuses on researchers and specialists, overlooking the need for practical guidance for industry practitioners who wish to apply them to specific tasks but lack specialized knowledge [48].

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
EASE '25, Istanbul, Turkiye
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1385-9/25/06
https://doi.org/10.1145/3756681.3756997

This paper is part of a research stream focusing on the operation challenges of MSA-based systems, which we have conducted over the last two years in collaboration with the industry. Here, we discuss the problem of linking the multitude of APIs forming these systems to the business functions they serve. We investigated this problem through an observational case study comprising three cases from the finance-insurance sector. Across them, operators dealt with an average of 725 APIs, approximately 40% of which were microservices. Their challenge illustrates the need to effectively handle API classification in the scope of fine-grained architectures.

We evaluated 14 decoder-only LLMs, encompassing general, code–, and domain-specific models, for classifying finance-insurance APIs based on their descriptions. We examined the effect of various prompt designs on classification performance using an expert-curated dataset (FinTechAPIs [21]). Additionally, we assessed the reliability of the LLMs by measuring output variability, crucial for mission-critical applications requiring consistency [9]. Our experiments were designed from the perspective of operators and SREs who improve their processes through LLM utilization, focusing on users unfamiliar with LLM development and operation, which reflects the profiles of practitioners observed in the three cases.

Our results demonstrate that general LLMs can correctly classify APIs without specific fine-tuning, with the best performer achieving an *F1–Score* of 89.5, with low output variability and high agreement, showing that the industry can adopt LLMs to such mapping with minimal effort, notably without requiring LLM expertise. This practical and actionable insight sets our work apart and highlights its relevance to real-world challenges. Its design and open-source tools offer a replicable framework for assessing LLM and prompt blending in SE classification tasks, while its quantitative analysis, covering performance, reliability, and response delays, provides objective guidelines for LLM-prompt selection based on specific goals. Ultimately, we evaluated the LLM-based solution from the practitioner's perspective, identifying persistent barriers to adoption and informing future research directions.

The remainder of this paper is organized as follows. Section 2 provides background and related work on LLMs and their application in API classification. Section 3 presents the industrial case studies. Section 4 details our research design. Section 5 presents the research results. Section 6 discusses findings and offers recommendations and perspectives. Section 7 discusses limitations. Finally, Section 8 concludes with remarks and future research outlook.

2 Background and Related Work

This section presents LLMs and their contributions to API-related tasks, as well as related work.

2.1 Large Language Models

LLMs are a family of machine learning (ML) algorithms designed to understand, generate, and predict human languages. They are trained on a superior amount and diversity of data compared to pioneer language models (LMs) [17]. Their size and complexity are often described in terms of the number of their parameters, ranging from millions to trillions [3]. Parameters enable the LM to capture advanced intricate patterns and nuances in the data but require significant computational resources for training and inference [30].

LLMs are classified into various families: encoder-only, encoder-decoder, and decoder-only [34]. Since their rise in 2022, decoder-only LLMs have gained momentum [14]. They use only the decoder module to incrementally build the output by sequentially predicting tokens from an initial state, handling downstream tasks without long and complex instructions [14, 34], and being able to grasp hidden information from the provided input, even if it is limited to a few words [2, 42]. While not exhibiting perfect reliability, LLMs have been explored in critical scenarios [40], making them potentially suitable to support mission-critical SE tasks.

LMs process SE artifacts similarly to other text types, adapting NLP techniques to SE tasks [49], such as code generation [1] and understanding [29], issue classification [6], and automatic documentation [10]. However, LLMs' accuracy may be limited by their ability to understand specific SE and development semantics, e.g., programming metaphors and idioms, as well as domain-specific information related to SE artifacts [15].

Training or fine-tuning an LLM faces challenges. First, the scarcity of curated data covering specific knowledge is critical to training specialized models [38]. Second, LLMs fine-tuned on particular tasks using compatible datasets are expected to outperform general ones, but training such an LLM faces high-quality domain dataset scarcity [50]. Finally, the complexity and costs of building and operating LLMs can limit their practical adoption [48]. Furthermore, current literature has focused on SE development tasks, overlooking operational ones [14]. It has targeted ML practitioners, which may prevent non-experts from having the necessary knowledge to make informed decisions concerning LLM adoption and use [48].

2.2 Related Works

The classification approach presented in this paper has rarely been studied in previous research. Extensive literature has discussed supporting API retrieval for developers, with some suggesting labelling APIs to support their categorization within catalogues [7]. However, our study does not focus on retrieving APIs when designing a system, where LLM retrieval approaches based on interpreting natural language users' inputs have exhibited impressive results [25]. Instead, we address the research gap in the practical application issues of LLMs in an SE operation task, i.e., API classification by operators and SREs, focusing on the concerns of these practitioners.

Few studies have applied LLMs to API classification in this context. In [28], the authors experimented with BERT, GPT-3.5, and GPT-4 using two datasets comprising 17 and 923 API descriptions from public API repositories. They extracted keywords from the API description with BERT and then populated a prompt sent to GPT-3.5 and GPT-4 for class selection. While they noted output variation in GPT-3.5 and GPT-4, they did not systematically measure them or discuss prompt design. They also highlighted quality issues in the experiment dataset. Similarly, only a few works investigated the impact of output variability on the consistency of LLMs ([4, 8]), primarily focusing on the OpenAI GPT family and lacking guidelines for industry adoption.

3 Industrial Cases

The three cases, presented in Table 1, have developed and operated complex software systems for over thirty years. They have

migrated their legacy systems to cloud microservices, resulting in the proliferation of APIs, as observed in other industries [51]. Their microservices APIs comprise between 1 and 20 endpoints and are documented using the OpenAPI standard [16], but their number increases constantly, resulting in a significant mismatch between the operators' knowledge base and the current microservices APIs running in production. The three cases manually built a unified view of systems based on their relation to business functionalities to support operations. Due to confidentiality concerns, we cannot disclose extensive details about these cases. Some information can be disclosed upon request to support reproducing this study.

Table 1: Case Study Description

Case	Industry	IT Teams	Operators	APIs
A	Banking	42	23	817
B	Health Insurance	36	16	673
C	Real Estate & Trading	38	13	685

4 Research Design

Our research design is based on guidelines for conducting observational case studies proposed in [46] and empirical studies proposed in [47], following the same rationale as in [12], which are adapted to research in real-world setups. We executed our research in six phases. Fig. 1 depicts our research design, which is detailed below.

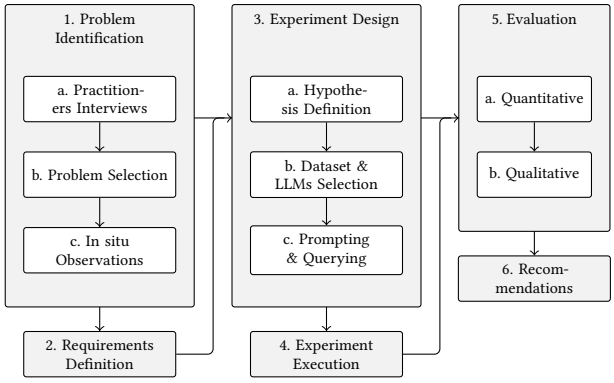

Figure 1: Research Design.

4.1 Problem Identification

Two researchers conducted an informal interview round with two practitioners from each case study (cf. Table 2 for participants' demographics). They were invited to answer open questions about their operation practices.[1] We recorded these interviews and relied on the automatic transcription facility of the recorder.[2] We reviewed the transcripts and fixed the errors. From their responses, we identify trends, shared goals, and concerns. Following this, we listed problems to be addressed. We submitted it to participants during a focus group session, where they collectively identified the linkage of business functionalities to APIs as a top concern, selecting it as the problem to be addressed.

Table 2: Participants Demographics.

ID	Case	Experience in Operations	Experience in SE	Role	LLMs Expertise
001	A	>15 y	>20 y	Senior Operator	Chatbot user
002	A	>10 y	>10 y	Senior Operator	Chatbot user
003	B	>5 y	>10 y	Operator	Chatbot user
004	B	>20 y	>25 y	Senior Operator	No expertise
005	C	>2 y	>10 y	SRE	Chatbot user
006	C	>10 y	>10 y	Senior Operator	Chatbot user

Then, the same two researchers conducted an *in situ* observation in Case A, from May to August 2024. At that time, Case A conducted a manual topology of their system, providing us with first-hand observation of their goals, processes and barriers they faced. First, we discussed the manual topology process with the operator who conducted it. Then, we explored Case A documentation and attended technical and follow-up discussions with supporting roles. Finally, we observed the process execution by doing four pair sections with the operator in charge of the topology, with the two researchers alternating participation. After these sections, the researchers compared their observations and requested clarification from the operator when their findings disagreed. Eventually, they compiled a final observation report. This in-situ observation taught us how practitioners achieve this task in real-world conditions.

4.2 Requirements Definition

The same two researchers conducted a second round of informal interviews with the same practitioners to identify the requirements and challenges they foresee in adopting an LLM-based API classification. None of the participants had trained or fine-tuned an ML model. Despite indicating they used AI-based tools, they exposed a lack of knowledge and practice in manipulating LLMs. For instance, the LLMs' *temperature* parameter was unknown to them. Nevertheless, all participants indicated that they were aware of the changes in responses provided by LLMs according to the "sentences used in questions," i.e., prompts.

Practitioners expressed concerns about the risks of data leakage when prompting LLMs, output variation, the infrastructure required to operate LLM-based solutions, and the associated costs, aligning with concerns raised by other industries [48]. Based on these outputs, we identified five industry requirements that an LLM-based solution should meet: reliable and constant classification, unrestricted on-premise installation or remote access, avoidance of security breaches, ease of use, and affordable costs. We did not consider cost-benefit, security, and ease-of-use requirements in this study because of the complexity of their assessment. From these observations, we derived the following research questions:

***RQ1.* How reliable is LLMs-based API classification?**

[1] cf. https://bit.ly/llms4apiclassification for interviews and focus group guides.
[2] Microsoft Teams

RQ2. **To what extent do recently released decoder-only LLMs expose output variation?**

RQ3. **Which aspects should be considered when deciding on the practical application of LLMs in APIs classification?**

4.3 Experiment Design

This section details the hypothesis, the dataset, the LLMs selection process, the prompt design, and queries.

4.3.1 Hypothesis Definition. The null hypothesis below and associated decision rules guided our experiments.

$H0_1$. None of the LLMs achieves sufficient accuracy in API classification.

$H0_2$. All domain-specific LLMs exhibit poor classification performance compared to general LLMs.

$H0_3$. Providing specific prompts to LLMs does not improve API classification performance.

$H0_4$. None of the LLMs exhibit high constancy in their output.

Rejecting $H0_1$ establishes LLMs' potential to accurately achieve the API classification task. Rejecting $H0_2$ means applying a domain-specific model improves API classification precision compared to general LLMs, reinforcing the assumption of a positive effect of task and domain-specific training on LLMs' performance. Rejecting $H0_3$ confirms that LLMs still exhibit sensitivity to prompt design. Rejecting $H0_4$ establishes that some last-generation LLMs are less subject to output variation. Simultaneously rejecting $H0_1$ and $H0_4$ evidences the practicability of adopting LLMs to API classification.

4.3.2 Dataset and LLMs Selection. We did not have access to real-world API documentation from the three cases for our experiments. Therefore, we selected a public dataset aligned with the case domain. The three cases support various financial products: bank accounts, multi-currency accounts, trading, currency exchange, payment and recovery, money transfers, loans and mortgages, health and goods insurance, and credit cards. However, there are insufficient high-quality human-curated datasets for API classification assessment[28], which limited our potential choices. Additionally, we avoided using the dataset employed by state-of-the-art baselines [28], as the authors explicitly noted concerns about the reliability of their results due to the dataset's poor annotation quality.

Therefore, we experimented with a subset of the FinTechAPIs dataset [21], an expert-annotated dataset of finance APIs constructed from open-source API descriptions extracted from public software repositories, achieving substantial agreement. This choice was intentional to ensure we evaluate the LLMs' capabilities within a controlled environment where structural inconsistencies in API descriptions would not confound our findings. The selected dataset is built from OpenAPI [16] documents, a widely adopted standard for API description [38], and comprises real-world API descriptions.

Each row in the dataset comprises an API description and a label, the classification target. Fig. 2 depicts the structure of an API description in FinTechAPIs. From the 249 API descriptions in this dataset, we excluded those labelled as "other" or "NO_CONSENSUS" and included only those where the agreement between annotators was unanimous (3/3), resulting in a subset of 231 rows. Table 3 shows the categories distribution of our FinTechAPIs-based dataset.

```
TITLE : {title}
DESCRIPTION :
    {description}
ENDPOINTS :
    - {endpoint1}
      {description and/or summary}
    - {endpoint2}
      {description and/or summary}
    ...
```

Figure 2: Structure of an API Description in FinTechAPIs.

Table 3: Class Distribution of the FinTechAPIs Subset.

Category	Support
banking	41
payment	25
loan-mortgage	24
user-password	22
client	21
currency	20
savings	20
transfer	20
trading	19
blockchain	19
Total	231

We selected the LLMs based on practitioners' requirements and technical properties defined by us. We considered decoder-only pre-trained models released between 2023-12-01 and 2024-09-30 to ensure assessing recent models. Their providers must allow unrestricted on-premise installation or remote access through an API. LLMs to be installed must have a *safetensors* version to avoid security breaches. They must be compatible with the *Transformers* library and the target infrastructure, utilizing a maximum of 8 GPUs with 80GB HBM2E. The selected LLMs must comprise at least a general model, a fine-tuned model for code-related tasks, and a fine-tuned model for the finance domain. Additionally, at least one of the LLMs must be open-source and one closed-source to comply with guidelines for comprehensive experiments with LLMs [36].

To limit the scope of evaluated LLMs and ensure feasibility, we focused our selection on the top 30 ranked models[3] from two recognized LLMs leaderboards [5, 23], prioritizing those with the best average results. We selected the latest release and the highest number of parameters when multiple versions were available. Except for the finance models, which we selected from recently published works because they were not featured on these leaderboards. We assessed the pre-selected models (16) compatibility with the evaluation infrastructure through simple queries. We discarded those exhibiting runtime errors or taking more than 2 hours to generate an output.[4] The final set comprises 14 LLMs detailed in Table 4.

4.3.3 Prompting and Querying. Literature has broadly discussed the effect of prompt design on LLM outputs [9, 18, 36, 43, 50]. Therefore, we decided to experiment with state-of-the-art prompt design

[3] As of 2024-06-25.

[4] The detailed analysis is available at https://bit.ly/llms4apiclassification

Table 4: LLMs considered in our research

Model	Parameters	Type	Access	Fine-tuned	Base	Release/ Update	Knowledge Cutoff
Llama–3.1	70B	Open–source	Local	—	—	2024/07	2023/12
Gemma–2	27B	Open–source	Local	—	—	2024/06	2024/06
Mistral–Large–2	123B	Open–source	Local	—	—	2024/07	2024/04
Qwen2.5	72B	Open–source	Local	—	—	2024/09	2024/09
Phi–3.5–MoE	42B	Open–source	Local	—	—	2024/08	2023/10
DeepSeek–V2.5	236B	Open–source	API	—	—	2024/09	—
GPT–4o–Mini	—	Proprietary	API	—	—	2024/07	2023/10
Gemini–1.5–Flash	—	Proprietary	API	—	—	2024/05	2023/11
Qwen2.5–Coder	7B	Open–source	Local	Code	Qwen2.5	2024/09	2024/09
DeepSeek–Coder–V2 (0724)	236B	Open–source	API	Code	DeepSeek–V2	2024/07	2023/11
Artigenz–Coder	6.7B	Open–source	Local	Code	DeepSeek–Coder–6.7B	2024/04	—
WizardCoder–V1.1	33B	Open–source	Local	Code	DeepSeek–Coder–33B	2024/01	—
Finance–Llama3	8B	Open–source	Local	Finance	Llama3–8B	2024/06	—
FinanceConnect	13B	Open–source	Local	Finance	Llama2–13B	2023/12	—

techniques, including category descriptions (*CD*) [6, 24], chain-of-thought (*CoT*) [45], and few-shot prompting (*FS*) [2, 11], to measure the LLMs' sensitiveness to prompt design and identify the strategies that achieved the highest *F1–Score*.

A typical LLM prompt comprises three parts: *system*, *user*, and *assistant*. The *system* part provides context for the task and describes the model's expected behaviour [43]. The user's question would constitute the *user* section of the prompt, serving as the primary input of the task, the instruction. The model's answer is the *assistant* part of the prompt. It can be used, along with the *user* section, to provide a conversation history when querying the model, either to offer additional context or to illustrate proper responses to the user [18, 45]. This chat history is created through multiple simulated back-and-forth interactions between the *user* and the *assistant* [24].

user	{*Ins*, *CD+Ins*, *Ins+CoT*, *CD+Ins+CoT*}	
assistant	Understood. Please provide the API summary.	
user	{API summary example}	*FS*
assistant	{Response example}	
⋮	⋮	
user	{API summary}	

Figure 3: Prompt template

Since not all the evaluated LLMs were trained to support the *system* role, our prompt template, depicted in Fig. 3, relies solely on the *user* and *assistant* roles to ensure consistency throughout experiments. The first *user* section comprises the classification task instructions, combining the *Ins*, *CD*, and *CoT* prompt elements. After including a generic response from the model indicating it understands the task, we complete the prompt with either few-shot (*FS*) examples or the API description.

Base Instructions (Ins) are structured in a numbered list and enclosed within <instructions> tags, as shown in Fig. 4. In these instructions, the *user* asks the model to read, analyze, and classify the provided API description and to return the category in a specific format. ***Category description (CD)*** gives the LLM definitions for the categories into which API documents should be classified, providing domain-specific and contextual data to the LLMs. Category descriptions are placed before the base instructions (*Ins*) and added inside <categories> tags (see Fig. 4). ***Chain of Thought (CoT)*** generates a series of intermediate reasoning steps (chain-of-thought) while answering user queries, which can significantly improve LLM's performance [18]. This part of the prompt instructs the model to output its rationale and selected category inside <thinking> and <category> tags, respectively. When incorporating the *CoT* approach in our prompt, we replace the fourth step of the base instructions (*Ins*).

Few-Shot Prompting (FS) involves including examples in the prompt to guide the model's responses and improve its understanding of the task by leveraging its in-context learning capabilities [2]. In our experiments, we included three examples of accurate API

CD	<categories> - banking: Traditional banking services, including account management, Automated Teller Machines (ATMs), credit card management and payment methods. - blockchain: Blockchain technology, such as cryptocurrency and smart contracts. - client: Client information management, including customer profiles, personal goals and credit rating. - currency: Currency exchange rates and currency conversion tools. - payment: Payment processing, including transactions, digital wallets and invoices. - savings: Financial planning tools for savings, investment plans, interest calculations and savings product. - trading: Trading activities, stock trading, forex trading, and investment portfolios. - transfer: Transferring funds between accounts, both domestically and internationally. - user-password: User authentication, password management, and security protocols (tokens) for user access. - loan-mortgage: Loan and mortgage processes, application submission and lenders. </categories>
Ins	<instructions> 1. You will receive an API summary. Read it carefully. 2. Identify the main functionality and purpose of the API. 3. Classify the API into one of the following categories: [banking, blockchain, client, currency, payment, savings, trading, transfer, user-password, loan-mortgage]. 4. Respond only with the category name. </instructions>
CoT	4. Write your thinking process in two sentences inside <thinking> tags. 5. Respond with the category name inside <category> tags.

Figure 4: Prompt elements

classifications through back-and-forth interactions between the *user* and *assistant* roles (see Fig. 3). Three examples are commonly used in LLM assessments to provide sufficient differentiation without exceeding the context window limit [41]. Thus, we randomly selected the three smallest API documents of each category to inform the LLM in cases where accessing limited data can make the classification decision difficult. By doing so, we intended to limit noise in the model's generated output, as it can produce superfluous data beyond the intended response format, adversely impacting classification performance. We excluded the documents used as examples from the testing dataset to ensure an unbiased evaluation.

We queried the LLMs using the following prompt combinations: base instructions (*Ins*), category descriptions and base instructions (*CD+Ins*), category descriptions, base instructions and few-shot prompting (*CD+Ins+FS*), base instructions and chain-of-thought (*Ins+CoT*), category descriptions, base instructions and chain-of-thought (*CD+Ins+CoT*), and all strategies (*CD+Ins+CoT+FS*). The expected output is a single label from the provided list that best describes the API. This label is extracted according to the output format specified in the prompt (i.e. the category alone or the category inside <category> tags).

4.4 Experiment Execution

Experiments ran from July to October 2024, updated with each new LLM release. We ran each pair of LLM-prompts five times over five days. We relied on two Python scripts, one for open-source models and the other for proprietary ones, both executing the same process: Initiating the experiment, gathering the results, and storing them in CSV files. Along with the classification results, we recorded runtime information, i.e., the duration of processing all the documents.[5] All tests involving local LLMs were run on a high-performance computing server with 1 TB of RAM and powered by eight NVIDIA A100 80GB GPUs. The scripts used Python version 3.11, the Transformers library version 4.43.1, and Pandas version 2.2.2, running in an Apptainer container.

4.5 Evaluation

Below, we detail the quantitative and qualitative evaluation.

4.5.1 Quantitative. We use the macro *F1–Score* metric to evaluate the performance of each LLM. *F1–Score* unveils how well the LLM identifies positive classes while minimizing false positives. The macro-averaging of this metric considers all classes equally, which is adequate for assessing models when we ignore the details of the dataset used during their training [39]. Additionally, we measured the average (*mean*) and standard deviation (*std*) of the *F1–Score* across multiple runs of the same experiments. These metrics provided a central performance value and an indication of performance variability, allowing us to objectively compare LLMs and prompts.

Output variation is a persistent issue in LLM use and research [4], which requires repeating experiments and evaluating output variation [36]. Thus, we measured output variation for each LLM-prompt combination over different runs. We relied on Krippendorff's alpha coefficient (k–α) [19] as the basic metric to derive the output variation rate. k–α typically measures the inter-rater agreement among independent annotators. Therefore, we considered each run of the LLM-prompt combination as an individual annotator. The coefficient ranges from 0 to 1, with values near 0 indicating high disagreement and significant output variation, while values close to 1 suggest greater agreement and less output variation. An undefined k–α makes assessing chance in agreement impossible, often occurring when an annotator labels all documents the same. For this study, we considered the LLMs exhibiting undefined k–α unreliable in achieving the task.

We employed specific criteria based on these metrics to validate the null hypotheses. Table 5 summarizes the rules applied for each hypothesis. We relied on Pareto's rule [33] as the threshold to establish LLMs and prompt performance based on the *F1–Score*, assuming high performance when achieving an *F1–Score* greater than or equal to 80. Considering the remaining 20 requires significant effort and knowledge, even for human experts.

Table 5: Validation Rules for the Null Hypotheses.

Hypothesis	Criterion	Rejection Rule
$H0_1$	*F1*	≥ 80
$H0_2$	*F1*	$\geq$ than general LLM's averaged *F1* OR of fine-tuned LLMs $\geq$ original LLM's *F1*
$H0_3$	*F1*	Any prompt combination with *F1* $\geq$ than *Ins* prompt
$H0_4$	k–α	≥ 0.80

We followed the guidelines provided in [26] to interpret the k–α: disagreement is systematic when < 0, no agreement is observed when equal to 0, poor when < 0.67, moderate between 0.67 and 0.79, acceptable when ≥ 0.80, and perfect when equal to 1.

4.5.2 Qualitative. We conducted two qualitative evaluation rounds. In the first round, we presented the experiment results to practitioners at Case A. In the second round, we presented the experiment results to a pool of practitioners from the three cases.

5 Results

This section presents our experimental results, highlighting LLM performance in API classification and showcasing the highest and lowest scores. We also reveal performance deviations between general, domain–, and task-specific models. Additionally, we discuss variations in LLM performance with different prompts and their output variability. Table 6 displays all results.

5.1 Overall Performance

We observed a mean *F1–Score* of 74.2 and a *std* of 22.3, measured from the best *F1–Score* reached by each LLM regardless of the prompt. Among the 14 evaluated LLMs, 11 models performed above this mean: Llama, Gemma, Mistral, Qwen, Phi, DeepSeek, GPT, Gemini, Qwen–Coder, DeepSeek–Coder, and Artigenz–Coder. In contrast, three models felt below this threshold: WizardCoder, Finance–Llama3, and FinanceConnect. The observed *std* indicates that models performed with significant variability, with the lowest *F1–Score* (6.10) reached by WizardCoder and the highest (89.5) by Gemini, thus a variation of ± 83.4 points.

[5] Code and generated files are available at https://bit.ly/llms4apiclassification

Table 6: Results

Model	*Ins*	*CD+Ins*	*CD+Ins+FS*	*Ins+CoT*	*CD+Ins+CoT*	*CD+Ins+CoT+FS*	Overall
Llama–3.1	76.1 ± 3.04 0.968	83.4 ± 3.67 0.972	85.3 ± 0.89 0.958	73.9 ± 0.99 0.910	74.8 ± 1.56 0.835	**85.5** ± 0.45 0.935	79.8 ± 5.43 0.930
Gemma–2	49.9 ± 0.00 1.000	67.7 ± 0.00 1.000	**80.6** ± 0.00 1.000	51.8 ± 0.00 1.000	74.5 ± 0.00 1.000	77.6 ± 0.00 1.000	67.0 ± 12.1 1.000
Mistral–Large–2	72.9 ± 0.00 1.000	73.4 ± 0.00 1.000	85.2 ± 0.00 1.000	72.4 ± 0.00 1.000	51.2 ± 0.00 1.000	**85.7** ± 0.00 1.000	73.5 ± 11.4 1.000
Qwen2.5	75.9 ± 0.40 0.968	86.0 ± 0.37 0.991	85.8 ± 0.61 0.987	78.7 ± 3.50 0.909	**87.0** ± 0.95 0.964	85.5 ± 0.82 0.972	83.2 ± 4.51 0.965
Phi–3.5–MoE	50.6 ± 0.00 1.000	74.3 ± 0.00 1.000	74.6 ± 0.00 1.000	71.0 ± 0.00 1.000	75.4 ± 0.00 1.000	**75.7** ± 0.00 1.000	70.3 ± 8.95 1.000
DeepSeek–V2.5	70.8 ± 0.46 0.977	83.7 ± 0.38 0.987	**86.8** ± 0.47 0.977	71.4 ± 0.63 0.856	71.5 ± 0.88 0.749	82.0 ± 3.26 0.965	77.7 ± 6.78 0.918
GPT–4o–Mini	71.6 ± 0.87 0.944	81.9 ± 3.12 0.966	82.9 ± 0.29 0.961	70.8 ± 0.76 0.882	79.6 ± 3.06 0.936	**86.2** ± 0.48 0.955	78.8 ± 6.04 0.941
Gemini–1.5–Flash	85.7 ± 3.30 0.981	**89.5** ± 0.44 0.981	89.1 ± 0.25 0.992	83.4 ± 3.28 0.925	89.4 ± 0.88 0.941	87.1 ± 3.15 0.948	**87.4** ± 3.22 0.961
Qwen2.5–Coder	60.2 ± 3.82 0.821	77.0 ± 1.35 0.900	71.6 ± 1.54 0.899	58.8 ± 1.60 0.776	73.5 ± 2.62 0.840	**81.5** ± 0.98 0.885	70.4 ± 8.64 0.854
DeepSeek–Coder–V2	72.1 ± 0.41 0.968	82.1 ± 0.39 0.969	81.3 ± 3.62 0.972	60.8 ± 0.74 0.718	13.6 ± 2.01 0.381	**85.0** ± 1.09 0.959	65.8 ± 24.7 0.828
Artigenz–Coder	6.74 ± 0.00 1.000	3.17 ± 0.00 1.000	73.3 ± 0.00 1.000	37.4 ± 0.00 1.000	9.23 ± 0.00 1.000	**77.0** ± 0.00 1.000	34.5 ± 30.9 1.000
WizardCoder–V1.1	0.00 ± 0.00 UND	0.00 ± 0.00 UND	0.00 ± 0.00 UND	3.12 ± 0.00 1.000	0.00 ± 0.00 UND	**6.10** ± 0.00 1.000	1.54 ± 2.34 1.000
Finance–Llama3	0.00 ± 0.00 UND	0.00 ± 0.00 UND	2.47 ± 0.66 0.325	40.3 ± 1.93 0.251	23.6 ± 3.12 0.140	**72.1** ± 1.11 0.817	23.1 ± 26.5 0.383
FinanceConnect	0.00 ± 0.00 UND	0.00 ± 0.00 UND	**40.3** ± 1.51 0.636	0.17 ± 0.33 UND	0.32 ± 0.39 -0.001	23.7 ± 2.39 0.428	10.7 ± 15.8 0.266

For each model, values on the first row are *F1–Score* mean and standard deviation. Values on the second row are k–α.

Llama, Gemma, Mistral, Qwen, DeepSeek, GPT, Gemini, Qwen–Coder, and DeepSeek–Coder exposed an *F1–Score* greater than or equal to 80, meeting the $H0_1$ rejection rule and demonstrating enough performance to be confidently applied to the task of API classification. In contrast, finance-specific models performed poorly, with the highest *F1–Score* of 72.1 for Finance–Llama3.

General LLMs demonstrated higher performance than their fine-tuned versions, with DeepSeek exhibiting an *F1–Score* of 86.8 and DeepSeek–Coder an *F1–Score* of 85.0, and Qwen exhibiting an *F1–Score* of 87.0 and Qwen–Coder an *F1–Score* of 81.5. Taking the mean of general LLMs' highest *F1–Score*s (84.6), only the DeepSeek–Coder reached this threshold among the domain and task-specific models. Thus, the $H0_2$ is rejected, as at least one domain-specific fine-tuned LLM exposed higher performance than the general LLM average.

5.2 Prompts Impact

We compared variation in the *F1–Score* between each prompt combination and the benchmark (the *Ins* prompt). Table 7 summarizes the results. We observed that the prompt *CD+Ins+CoT+FS* improved LLMs' performance by +22.7 points on average, with 21.1 *std.* This prompt positively impacted all models, with the highest impact on Finance–Llama3 (+72.1 points). In contrast, *CD+Ins+CoT* achieved an average improvement of 2.2 points and 20.7 *std.* This prompt negatively impacted three LLMs: Llama, Mistral, and DeepSeek–Coder, with the worst impact on DeepSeek–Coder (−58.4 points). Regarding variation magnitude (*spread*) of LLMs *F1–Score*, we observed that *CD+Ins+CoT* exhibited a considerable difference of 83.3 points between the highest improvement and decline of the *F1–Score*. In contrast, *CD+Ins* exhibited the lowest *spread*: 27.3.

Besides, five models (Mistral, DeepSeek–Coder, Artigenz–Coder, Finance–Llama3 and FinanceConnect) demonstrated high sensibility to prompt design, with a *spread* of more than 30 points according to the prompt used. Artigenz–Coder exposed the highest variation with a *spread* of 73.9, while Phi exposed the lowest: 4.65. It is noteworthy that models that performed the best (Gemini and Qwen) also exposed low sensibility to prompts, with a spread of 6.1 and 8.1, respectively. We observe the same with the model performing the worst: WizardCoder (*F1–Score* of 1.54), suggesting consistency in their performance despite the prompt design. These results support rejecting $H0_3$, as various prompt combinations positively impacted the LLMs' classification performance. Indeed, all models improved performance over the original *Ins* prompt. Nevertheless, the results also pointed out that some prompts may strongly affect LLM output, as five LLMs demonstrated high sensitivity to prompt changes.

5.3 Output Variation

We assessed output variation by analyzing the k–α values of LLM-prompt combinations across five runs. Regardless of the highest *F1–Score* achieved, thirteen models exposed a k–α greater than 0.8, with five demonstrating perfect agreement (k–α = 1). Among them, four exhibited no variability, outputting the same response for each run. From these five, only WizardCoder exhibited variability according to the prompt. FinanceConnect exhibited poor agreement, below 0.67. Also, the k–α of all LLMs exhibited a low standard deviation concerning the prompt, the highest being 0.276, indicating a low influence of the prompt on their agreements. Nevertheless, we observed systematic disagreement (k–$\alpha \leq 0$) for the

Table 7: *F1–Score* Variation against *Ins* Prompt

Model	*CD +Ins*	*CD +Ins +FS*	*Ins +CoT*	*CD +Ins +CoT*	*CD +Ins +CoT +FS*	Spread
Llama–3.1	7.35	9.24	-2.17	-1.26	9.44	11.6
Gemma–2	17.8	30.7	1.90	24.6	27.7	28.8
Mistral–Large–2	0.46	12.3	-0.48	-21.7	12.8	34.5
Qwen2.5	10.1	9.90	2.79	11.0	9.62	8.25
Phi–3.5–MoE	23.7	24.1	20.5	24.9	25.1	4.65
DeepSeek–V2.5	12.9	16.1	0.65	0.71	11.2	15.4
GPT–4o–Mini	10.2	11.3	-0.87	7.97	14.6	15.5
Gemini–1.5–Flash	3.82	3.37	-2.26	3.64	1.39	6.09
Qwen2.5–Coder	16.8	11.5	-1.40	13.4	21.4	22.8
DeepSeek–Coder–V2	9.97	9.17	-11.2	-58.4	12.9	71.3
Artigenz–Coder	-3.56	66.6	30.6	2.50	70.3	73.9
WizardCoder–V1.1	0.00	0.00	3.12	0.00	6.10	6.10
Finance–Llama3	0.00	2.47	40.3	23.6	72.1	72.1
FinanceConnect	0.00	40.3	0.17	0.32	23.7	40.3
Spread	27.3	66.6	51.6	83.3	70.7	
Improvement mean	7.8	17.6	5.8	2.2	22.7	
Standard deviation	7.84	17.3	13.8	20.7	21.1	

LLM-prompt combination FinanceConnect-*CD+Ins+CoT*. Based on these findings, we reject $H0_4$.

5.4 Practitioners Evaluation

We first presented our experiment results to operators (2) and managers (1) at Case A, then to 63 practitioners: operators (15), SREs (2), developers (31), data scientists (14), and managers (3) from the three cases. We focused on LLMs' reliability and ease of use. Afterward, attendees shared their impressions, unanimously agreeing on the potential of LLM-based API classification. However, five operators and six developers emphasized the need for greater LLM literacy to determine the optimal blend of model and prompt. One manager raised concerns about cost and sustainability, stressing the need for guidelines in decision-making. LLM security was highlighted by the two SREs and a developer, emphasizing the need for a balance between cost-effectiveness and security. One data scientist noted that implementing this approach on-premise requires specific expertise. A developer summarized these points:

> The proposal is appealing and helpful. However, I'm not able to design a project to implement it. I do not know how to evaluate the time, resources and costs to implement such a thing. Also, I cannot ensure my boss that our data will be kept safe and not shared with externals.

6 Discussion

Examining the extent of LLMs' reliability (RQ1) and comparing their output variation (RQ2) helps us decide whether they are suitable for the API classification task. Eight of the evaluated LLMs exhibited acceptable rates for both aspects. Although none of them achieved perfect alignment with the curated dataset, they exhibited individual agreement (k–α) superior to the one observed among the FinTechAPIs annotators (0.814 [21]). Therefore, these LLMs demonstrated enough reliability to be considered a viable solution to support API classification. From these insights, we propose the following recommendations (**R**) and perspectives (**P**).

R-1: Practitioners should assess the expected reliability level and recognize that perfect matching may be unreachable.

P-1: Improving reliability could be achieved by applying LLMs in a pairwise classification approach [13], where they suggest a class and inform the human to the final decision. This approach should detail the LLM's decision rationale, which can be achieved with the *CoT* prompt.

The decision on the practical application of LLMs encompasses technical, economic, and security aspects (RQ3). Open-source LLMs grant users control of operational elements, including building and operating the infrastructure where the LLM will run, which can lead to significant challenges and increased costs [37]. Handling LLM operations also ensures data control, limiting the risk of data breaches. Some evaluated open-source LLM editors provide outsourced platforms (e.g., DeepSeek), allowing users to query the LLMs remotely. This option avoids needing an on-premise infrastructure but comes with data security risks. Closed-source LLMs avoid needing an on-premise infrastructure as they are accessed remotely through APIs. However, they have costs and expose users to the same risks [37].

R-2: Practitioners should evaluate their tolerance for data breaches and the LLM's compliance with industry standards and regulations on data protection.

R-3: Practitioners should assess their ability to manage the necessary infrastructure, considering costs and expertise.

R-4: Practitioners should assess LLMs' execution costs when outsourcing, as providers' plans vary significantly.

P-2: Practitioners might adopt an LLM federated approach to mitigate security issues [20].

Finally, practitioners should consider the time required for document processing. We observed variability among LLMs, with some models being faster than others. Reducing processing time is a key aim, so this factor must be carefully considered. We provide a summary in Table 8 of the best-performing LLMs and prompts, detailing their performance, output variability, and execution time (ET) for processing 231 documents.

R-5: Practitioners should compare their data volume to the average LLM processing capacity to assess the processing time required.

Table 8: Summary of Highest Performant LLMs

Model	Prompt	*F1–Score*	k–α	ET
Gemini–1.5–Flash	*CD+Ins*	89.5	0.981	3 min
Qwen2.5	*CD+Ins+CoT*	87.0	0.964	23 min
DeepSeek–V2.5	*CD+Ins+FS*	86.8	0.977	5 min
GPT–4o–Mini	*CD+Ins+CoT+FS*	86.2	0.955	35 min
Mistral–Large–2	*CD+Ins+CoT+FS*	85.7	1.000	49 min
Llama–3.1	*CD+Ins+CoT+FS*	85.5	0.935	29 min
DeepSeek–Coder–V2	*CD+Ins+CoT+FS*	85.0	0.959	19 min
Qwen2.5–Coder	*CD+Ins+CoT+FS*	81.5	0.885	7 min

Based on this summary, we can rank the LLMs demonstrating superior efficacy in classifying APIs. The most performant was Gemini, *F1–Score* of 89.5, followed by Qwen, 2.5 points behind, both proprietary LLMs. The most performant open-source LLM was DeepSeek, 2.7 and 0.2 points behind Gemini and Qwen. Only

DeepSeek–Coder exhibited superior efficacy among the domain-specific models, 4.5, 2, and 1.8 points behind Gemini, Qwen, and DeepSeek, respectively. Therefore, the deviation between them must be more significant to accept that fine-tuning LLMs with a task or domain-specific knowledge is no longer necessary for similar classification tasks. However, our findings demonstrated that, for API classification, general LLMs outperformed domain-specific ones, achieving the task without additional domain-driven fine-tuning. As these models have not been trained on a specific task or data from a single domain, we expect them to yield similar results when applied in a different domain.

This finding supports our hypothesis that current LLMs can be used as-is for effective API classification, providing grounds for selecting the adequate LLM that meets organizations' concerns and requirements. The outputs of our experiments can support decisions on the LLM precision (*F1–Score*), output reliability (k–α), access mode (local vs. remote), source type (proprietary vs. open-source), prompt complexity (simple vs. complex) and response delay (fast response or significant delay for providing an output), according to adopter's specific conditions. Besides, the experiment design can be replicated and extended to other LLMs, offering a framework to collect data and support the selection process.

7 Threats to Validity

This study has limitations and threats to validity that must be considered when interpreting its results [47]. LLMs experiments are suggested for specific threats for which comprehensive and proven mitigation strategies have yet to be built [36]. Here, we disclose these threats and outline the mitigation strategies adopted.

Construct validity refers to how well a test measures the concept it is intended to measure. A significant threat to construct validity in our study is the variability of LLM outputs, which impacts our ability to measure its constant performance in the API classification task. We took statistical measures for each LLM run and calculated their average to mitigate this threat. We also relied on a statistical measure to evaluate the regularity of the models' outputs (k–α), considering each run as an independent actor. By addressing this threat, we ensured fair conclusions.

Internal validity involves factors affecting experiment outcomes unknowingly to the researcher. We identify four LLMs-specific internal validity threats: unadapted prompt design, implicit data leakage, configuration, and time-based output drift [36]. An unadapted prompt design occurs when the prompt fails to align with the model's characteristics. Connecting an LLM output to a specific prompt is a complex task not covered in this study. As a mitigation, we used current prompt design practices in classification tasks [6], using task-specific and contextual prompts [45].

Implicit data leakage occurs when researchers cannot ensure that the evaluation dataset has not been used for training. We reviewed LLM documentation and associated publications, finding no comprehensive list of datasets used in their training. Thus, it is uncertain whether the experimented LLMs have previously accessed parts of FinTechAPIs. Although the dataset is new, it has been compiled from publicly available documents that these models may have accessed. Changing LLMs' parameters may affect output consistency, potentially leading to unfair comparisons. To mitigate this risk, we established a unique configuration that we applied uniformly. The time-based output drift occurs when the model output changes over time. As a mitigation, we ran each experiment five times for each model over five days and measured the LLM's average performance and output variability [36]. Moreover, this threat can arise from model evolution. We mitigated this risk by running the open-source models locally, avoiding updates. This strategy does not apply to closed-source models, as we cannot control their evolution. Thus, we can only specify the LLM version and conduct the experiments with a short delay to limit the impact of changes.

Our results may only generalize within the specific studied LLM versions, datasets, configurations and prompts. Further investigation is needed to validate our results beyond our study's setup. We provide a package containing our experiments' input and output artifacts and the replicable infrastructure. However, LLMs are non-deterministic and can produce different outputs for the same input [4, 36], even if the model temperature is set to 0 [32]. Therefore, we cannot ensure that reproducing our experiment will yield the same classification results. Nevertheless, we documented the observed output variations, which can serve as comparison points to assess the reliability of reproductions.

8 Conclusion

This paper presented an exploratory study of applying LLM-based classifiers for API classification. We conducted several experiments to assess the individual capabilities of these models. Besides, we evaluated the perspectives of LLMs in supporting SE operation tasks requiring reliable and comprehensive knowledge mining and processing. Our findings indicate that these models achieve this task with significant reliability, with an *F1–Score* of 89.5 and a k–α of 0.981 for the most performant model-prompt combination. Our results also highlight the importance of prompts in the models' performance. The different compositions of LLMs-prompts we experimented with can serve as starting points for adopting LLMs-supported API classification. Moreover, our findings establish a foundation to support decisions by non-experts when adopting such an LLM-based classification approach for similar tasks. Our future research will focus on experimenting with optimization strategies, allowing us to establish the most suitable blend of LLMs, prompts, and infrastructure. Additionally, we plan to extend our experiment to include other models.

References

[1] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B Ashok, and Shashank Shet. 2024. Codeplan: Repository-level coding using llms and planning. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 675–698.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 159.

[3] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? *arXiv preprint arXiv:2307.09009* (2023).

[5] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI]

[6] Giuseppe Colavito, Filippo Lanubile, Nicole Novielli, and Luigi Quaranta. 2024. Leveraging GPT-like LLMs to Automate Issue Labeling. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 469–480.

[7] Leonardo da Rocha Araujo, Guillermo Rodríguez, Santiago Vidal, Claudia Marcos, and Rodrigo Pereira dos Santos. 2021. Empirical Analysis on OpenAPI Topic Exploration and Discovery to Support the Developer Community. *Computing and Informatics* 40, 6 (2021), 1345–1369.

[8] Joost CF de Winter. 2023. Can ChatGPT pass high school exams on English language comprehension? *International Journal of Artificial Intelligence in Education* (2023), 1–16.

[9] Matteo Esposito and Francesco Palagiano. 2024. Leveraging Large Language Models for Preliminary Security Risk Analysis: A Mission-Critical Case Study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 442–445. https://doi.org/10.1145/3661167.3661226

[10] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[11] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. 2024. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. In *46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[12] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. 2006. A model for technology transfer in practice. *IEEE software* 23, 6 (2006), 88–95.

[13] Trevor Hastie and Robert Tibshirani. 1997. Classification by pairwise coupling. *Advances in neural information processing systems* 10 (1997).

[14] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).

[15] Mia Mohammad Imran, Preetha Chatterjee, and Kostadin Damevski. 2024. Shedding Light on Software Engineering-specific Metaphors and Idioms. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[16] OpenAPI Initiative. 2021. OpenAPI Specification v3.1.0. https://spec.openapis.org/oas/latest.html.

[17] Katikapalli Subramanyam Kalyan. 2023. A survey of GPT-3 family large language models including ChatGPT and GPT-4. *Natural Language Processing Journal* (2023), 100048.

[18] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[19] Klaus Krippendorff. 2018. *Content analysis: An introduction to its methodology*. Sage publications.

[20] Jahnavi Kumar and Sridhar Chimalakonda. 2024. Code Summarization without Direct Access to Code - Towards Exploring Federated LLMs for Software Engineering. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. 100–109. https://doi.org/10.1145/3661167.3661210

[21] Edwin Lemelin, Gabriel Morais, and Oumeima Nuigues. 2024. *The Fin-Tech API Descriptions Dataset (FinTechAPIs)*. https://github.com/UQAR-TUW/FinTechAPIsDataset

[22] James Lewis and Martin Fowler. 2014. Microservices: a definition of this new architectural term. *MartinFowler. com* 25, 14-26 (2014), 12.

[23] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* 36 (2024).

[24] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[25] Zexiong Ma, Shengnan An, Bing Xie, and Zeqi Lin. 2024. Compositional API Recommendation for Library-Oriented Code Generation. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. 87–98.

[26] Giacomo Marzi, Marco Balzano, and Davide Marchiori. 2024. K-Alpha Calculator–Krippendorff's Alpha Calculator: A user-friendly tool for computing Krippendorff's Alpha inter-rater reliability coefficient. *MethodsX* 12 (2024), 102545.

[27] Gabriel Morais, Mehdi Adda, and Dominik Bork. 2024. Breaking Down Barriers: Building Sustainable Microservices Architectures with Model-Driven Engineering. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 528–532.

[28] Gabriel Morais, Edwin Lemelin, Mehdi Adda, and Dominik Bork. 2024. Enhancing API Labelling with BERT and GPT: An Exploratory Study. In *EDOC 2024: 28th International Conference on Enterprise Design, Operations, and Computing – Forum Paper*. Springer.

[29] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[30] Peter Norvig and Stuart Russell. 2021. Artificial intelligence: a modern approach. *Pearson, Harlow* 1 (2021), 1239–1269.

[31] Joshua Ofoeda, Richard Boateng, and John Effah. 2019. Application programming interface (API) research: A review of the past to inform the future. *International Journal of Enterprise Information Systems (IJEIS)* 15, 3 (2019), 76–95.

[32] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2024. An empirical study of the non-determinism of chatgpt in code generation. *ACM Transactions on Software Engineering and Methodology* (2024).

[33] Vilfredo Pareto. 2014. *Manual of political economy: a critical and variorum edition*. OUP Oxford.

[34] Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. 2024. A review on large Language Models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access* (2024).

[35] Navid Sahebjamnia, S Ali Torabi, and S Afshin Mansouri. 2015. Integrated business continuity and disaster recovery planning: Towards organizational resilience. *European journal of operational research* 242, 1 (2015), 261–273.

[36] June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the silence: the threats of using llms in software engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 102–106.

[37] Glorin Sebastian. 2023. Privacy and data protection in ChatGPT and other AI Chatbots: strategies for securing user information. *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)* 15, 1 (2023), 1–14.

[38] Souhaila Serbout and Cesare Pautasso. 2024. APIstic: a large collection of OpenAPI metrics. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 265–277.

[39] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information processing & management* 45, 4 (2009), 427–437.

[40] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine* 29, 8 (2023), 1930–1940.

[41] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2024. Language models don't always say what they think: unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems* 36 (2024).

[42] Yao Wan, Zhangqian Bi, Yang He, Jianguo Zhang, Hongyu Zhang, Yulei Sui, Guandong Xu, Hai Jin, and Philip Yu. 2024. Deep Learning for Code Intelligence: Survey, Benchmark and Toolkit. *Comput. Surveys* (2024).

[43] Li Wang, Xi Chen, XiangWen Deng, Hao Wen, MingKe You, WeiZhi Liu, Qi Li, and Jian Li. 2024. Prompt engineering in consistency and reliability with the evidence-based guideline for LLMs. *npj Digital Medicine* 7, 1 (2024), 41.

[44] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. 2020. A systematic mapping study on microservices architecture in devops. *Journal of Systems and Software* 170 (2020), 110798.

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[46] Roel J Wieringa. 2014. *Design science methodology for information systems and software engineering*. Springer.

[47] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. 2012. *Experimentation in Software Engineering*. Springer. https://doi.org/10.1007/978-3-642-29044-2

[48] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data* 18, 6 (2024), 1–32.

[49] Yanming Yang, Xin Xia, David Lo, and John Grundy. 2022. A survey on deep learning for software engineering. *ACM Computing Surveys (CSUR)* 54, 10s (2022), 1–73.

[50] Shangbo Yun, Shuhuai Lin, Xiaodong Gu, and Beijun Shen. 2024. Project-specific code summarization with in-context learning. *Journal of Systems and Software* (2024), 112149.

[51] Xin Zhou, Shanshan Li, Lingli Cao, He Zhang, Zijia Jia, Chenxing Zhong, Zhihao Shan, and Muhammad Ali Babar. 2023. Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. *Journal of Systems and Software* 195 (2023), 111521.