Enhancing API Labelling with BERT and GPT: An Exploratory Study

Gabriel Morais[1], Edwin Lemelin[1,2], Mehdi Adda[1], and Dominik Bork[1,3]

[1] Université du Québec à Rimouski, Lévis, QC, G6V 1L8, Canada
{gabrielglauber.morais,edwin.lemelin, mehdi_adda}@uqar.ca
[2] Université Laval, Québec, QC, G1V 0A6, Canada
[3] Business Informatics Group, TU Wien, Vienna, Austria
dominik.bork@tuwien.ac.at

Abstract. Application Programming Interfaces (APIs) enable interaction, integration, and interoperability among applications and services, contributing to their adoption and proliferation. However, discovering APIs has relied on manual, time-consuming, costly processes that jeopardize their reuse potential and accentuate the need for effective API retrieval mechanisms. Leveraging the OpenAPI Specification as a basis, this paper presents an exploratory study that combines BERT and GPT machine learning models to propose a novel API classifier. Our investigation explored the *zero-shot learning* capabilities of GPT–4 and GPT–3.5 using relevant terms extracted from API descriptions using BERT. The evaluation of our approach on two datasets comprising 940 API descriptions sourced from public repositories yielded an F1-score of 100% in the small dataset (17 APIs) and 39.1% in the large dataset (923 APIs). These results surpass state-of-the-art on the small dataset with an impressive 29-point improvement. The large dataset showed GPT can suggest labels not in the provided list. Manual analysis revealed that GPT's suggested labels fit the API intent better in 18 out of 20 cases, highlighting its potential for unknown classes and mismatch detection. This emphasizes the need to improve dataset quality and availability for API research. Our findings show the potential of automated API retrieval and open avenues for future research.

Keywords: API classification, OpenAPI Specification, GPT, BERT.

1 Introduction

API descriptions are essential in software engineering as they serve as documentation [7], enabling developers to understand and interact with them effectively [21]. Acting as a contract between different components, they ensure seamless integration and rapid development. In this context, the OpenAPI Specification (OAS) [15] standard provides a machine-readable format for describing RESTful APIs, allowing automation capabilities, e.g., code generation [7].

Getting a general understanding of an API is the most critical task when mining API for use in specific contexts [7, 21]. In most cases, this task is executed manually [10], and the number of APIs to explore could impede its completion.

The first step toward an effective retrieval mechanism is organizing and classifying data. Automatic text classification is a long-standing research field [18], and recent advances in machine learning, e.g., large language models (LLM), have opened up new avenues to increase the accuracy of automated text classification approaches [4].

Automatically processing OpenAPI documents (OADs) could enhance API discovery and cope with time-consuming manual exploration. Nevertheless, OADs are specific textual documents that combine structured data with a blend of general and technical lexicons. Previous works have explored using OADs and natural language processing (NPL) techniques to improve API discovery [33, 23, 35]. However, the potential of labelling APIs from OADs using current state-of-the-art LLM algorithms is a significant area yet to be fully explored, with the potential to make a substantial impact in the field.

In this exploratory study, we present preliminary results of applying GPT–4 and GPT–3.5 in combination with KeyBERT to label OADs. The explored approach used KeyBERT to extract relevant words from OADs, which were then passed to GPT through a prompt to find the corresponding label of the document. We relied on KeyBERT to help us enhance prompt engineering, cope with prompting payload restrictions, and limit costs related to OpenAI API consumption.

We experimented with a small labelled dataset comprising 17 OADs used in Microservices research [3]. Our approach achieved an F1-score of 100 in this dataset, an improvement of 29 points from previous works [23]. Encouraged by these results, we assessed the approach's generalizability using a dataset of 923 OADs from the APIs.Guru repository [2] and achieved an F1-score of 39.1.

The experiments further revealed that data cleaning is not required, as words extracted by KeyBERT could be used as-is. Similarly, a manual analysis of labelling mismatches showed that 18 out of the 20 analyzed mismatches were caused by inappropriate labels on the *apisguru* dataset, unveiling the need for accurate datasets to support machine-learning approaches in API labelling research, and new application perspectives, e.g., applying GPT to mislabelling detection. All the artifacts used and produced during our experiments and evaluation are available at this paper's code companion [24].

The remainder of this paper is organized as follows. Section 2 provides the essential background to understand the proposed approach. Section 3 introduces our approach. Section 4 presents the results obtained. These results are discussed in Section 5, along with the impacts of our study. Finally, Sections 6 and 7 provide the future research agenda and concluding remarks, respectively.

2 Background

This section introduces concepts that support readers' understanding of the proposed approach. First, we present the OpenAPI Specification applied to API descriptions (2.1). Then, we provide a high-level description of the Transformer

machine learning architecture (2.2), providing basic knowledge to understand how BERT and GPT models work.

2.1 OpenAPI Specification

The OpenAPI Specification (OAS) [15], formerly known as Swagger Specification, is a formal specification defining a standard to describe, document, and communicate RESTful APIs. OAS-based API descriptions are formalized hierarchically using a key-value approach, where the keys are the OAS parameters, ensuring consistency throughout different OAS-based API descriptions [30]. These descriptions are typically serialized in YAML Ain't Markup Language (YAML) or JavaScript Object Notation (JSON), which makes them machine-readable.

OAS includes general information about the API, such as its paths, operations, input and outputs, and security details. The values associated with OAS keys can be in any language and follow any standard. Often, they are represented as natural language terms using programming languages' conventions [7]. Listing 1 shows an excerpt of an API described using OAS in YAML.

While the OpenAPI Specification provides various benefits, its effectiveness depends on the accuracy and diligence of the individuals creating the API descriptions [13]. Human errors, oversight, or lack of attention to detail during the API documentation process can result in inaccurate or incomplete descriptions, leading to discrepancies between the documented API and its actual implementation. This risk of unreliability is not limited to using OAS for API descriptions but seems inherent to API documentation [37].

2.2 Transformers-based ML Models

Transformer is a deep learning architecture built on the attention mechanism and comprising two main layers: *Encoders*, which create a representation of the input data, and *decoders*, which generate the output sequence step by step [38]. These components enable the Transformer architecture to handle sequence-to-sequence tasks, such as machine translation, text classification, and generation.

Transformer *encoders* and *decoders* are almost similar; both have a self-attention mechanism and a feed-forward network. However, decoders have an additional sub-layer that applies self-attention to encoders' outputs. The self-attention mechanism captures long-distance context without a sequential dependency, allowing each position in the sequence to attend to other positions, capturing long-range dependencies [25]. This additional self-attention mechanism copes with *decoders* limitation of accessing only previous positions. Sequence ordering is captured through a positional embedding, which is added to the input embeddings in the encoder and decoder stacks to provide information about the sequence's relative or absolute tokens' position. Mainly, encoders are helpful for text classification tasks, while decoders are helpful for text generation tasks.

Generative Pre-trained Transformer (GPT) [31] is a state-of-the-art deep learning model for NLP tasks built upon *decoders*. It is trained in two steps,

```yaml
openapi: 3.0.1
info:
  title: Checkout API
  version: v1
paths:
  "/api/v1/Checkout":
    get:
      tags:
      - Checkout
      parameters:
      - name: userName
        in: query
        schema:
          type: string
          nullable: true
      responses:
        '200':
```

Listing 1: EShopOnContainers Checkout API's OAD

```
swagger
api
apis
json
yaml
apisguru
openapi
$ref
ref
jsonschemadialect
servers
paths
webhooks
components
security
tags
...
```

Listing 2: Ignored terms

first by unsupervised generative pre-training of a language model using a massive amount of unlabelled text data, then by a supervised discriminative fine-tuning of the pre-trained model on specific downstream tasks. During fine-tuning, the model adapts its learned representations to suit the task at hand, enabling it to improve performance on various NLP benchmarks.

GPT–3 is a significant improvement of the GPT series that introduced a 175 billion parameters model, 100 times bigger than its predecessor (GPT–2). GPT–3 relies on the model size to learn more from diversified sources, resulting in models that can achieve various tasks without task-specific training, starting the era of large language models (LLMs) [16]. GPT–3 allows users to bypass the supervised fine-tuning of the pre-trained model by directly providing instructions during inference, such as a task description and output examples [6]. This approach is called *in-context learning*. It dramatically reduces fine-tuning efforts without losing the model's accuracy. GPT–4 is the last evolution of the GPT series, comprising 170 trillion parameters and supporting text and image inputs [1]. OpenAI made GPT–3 and GPT–4 available through an API, allowing developers and businesses to access and utilize the models' capabilities on a broader scale, contributing to their popularity and hype.

Bidirectional Encoder Representations from Transformers (BERT) is an NLP model composed of 340 million parameters and trained on an extensive dataset of 3.3 billion words [9]. It is considered a state-of-the-art technique in various NLP tasks [22]. It employs a multi-layer Transformer-Encoder architecture with self-attention mechanisms and feed-forward neural networks, following a two-step process of pre-training and fine-tuning. The pre-training process re-

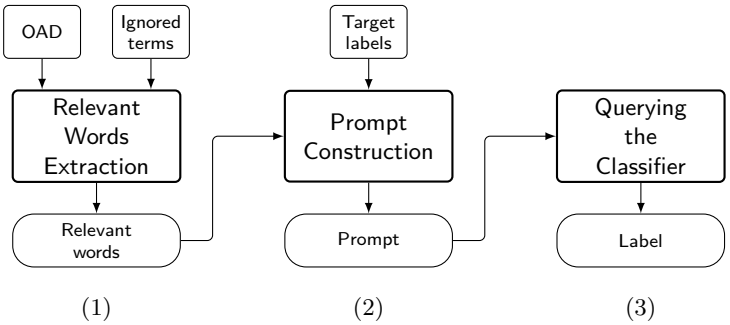

Fig. 1: Overview of our BERT and GPT-based API Labelling Approach

lies on a masked language modelling task, where specific tokens are randomly masked. The model then recovers these masked tokens by considering the encoding vectors from a bidirectional Transformer, allowing it to understand the context from both the left and right sides. The fine-tuning process is achieved by training with reduced resources and smaller datasets to optimize its performance for specific tasks. BERT comes in different sizes and has variants called the BERT family [22].

In summary, each model has strengths and limitations. The choice of the model depends on the problem, use case, and the available data. Recently, the combination of GPT and BERT has mutually improved their abilities to solve question-answering tasks [17]. This work inspired our approach.

3 Approach

Our approach, as depicted in Fig. 1, offers a comprehensive solution for API classification. Given an OpenAPI document (OAD), a list of terms to be ignored, and a list of target labels, our approach proceeds through a series of systematic steps to achieve classification. First, we extract a specific number of relevant words from the API description while excluding OAS terms and English stopwords from the provided exclusion list (1). Next, we build a classification prompt using the extracted relevant words and a list of target labels (2). Finally, the classifier is queried using the generated prompt, assigning the OAD to its corresponding label (3).

3.1 Relevant Words Extraction

To extract relevant words from an OpenAPI Document, it initially needs to be parsed from YAML or JSON and converted into a single string of lowercase words. Then, unique words within the text that are not in the list of terms to ignore are counted. This list comprises common English stop words and OAS key names (see Listing 2). For recall, API documents formalized using the OpenAPI

```
system : "You will receive keywords extracted from OpenAPI documents.
Your task is to classify the document into one of the following
categories: {target_labels}.
Respond only with the category name."
user : "{relevant_words}"
```

Listing 3: GPT Prompt Template

standard follow a key-value approach, with keys defined within the specification, leading to the recurrence of these terms (keys) in OADs.

As an illustrative example, by ignoring these words, the OAD excerpt from Listing 1 contains nine unique words: `cancel`, `eshoponcontainers`, `header`, `info`, `ordering`, `orders`, `requestid`, `service`, and `v1`. The number of relevant words to extract is determined as 20% of the total count of unique terms in the document, following Pareto's principle [34], also known as the 80/20 rule, which states that roughly 80% of the effects come from 20% of the causes. Thus, we considered that 20% of the OAD content is enough to characterize it.

Keywords are retrieved using KeyBERT [11], a Python package that leverages BERT word embeddings. It extracts terms or sentences that are most similar to the document based on cosine similarity. Using KeyBERT, we provide the number of words to extract and the list of terms to exclude, as shown in Listing 2. This ensures that the keywords obtained are significant. Once the process is finished, these words are compiled into an enumeration for use in the prompt.

3.2 Prompt Construction

The large language model prompt used to classify OADs is constructed by combining the relevant words extracted with KeyBERT and user-defined target labels. The prompt comprises two parts: `system` and `user`. The motive for this segmentation is explained in the following Subsection. We use basic string concatenation techniques to incorporate the user's input in this configurable prompt. Listing 3 presents the template used in our approach. The characters in black are fixed, while those in orange and magenta correspond to the variable names of target labels and relevant words, respectively. This template has proven to work best in our testing.

3.3 Querying the Classifier

Both the `system` and `user` segments of the prompt are sent to GPT using OpenAI's API and its chat completion endpoint. The `system` message conveys high-level information or context to guide the model's behaviour. In contrast, the `user` part contains the direct inputs from the users, representing prompts they want the model to respond to [28]. Thus, the GPT API is queried using this prompt, and the response is trimmed and converted to lowercase. Only the first label

```
system : "You will receive keywords extracted from OpenAPI documents.
Your task is to classify the document into one of the following
categories: audit, rooting, cart, frontend, order, catalog, user,
currency, location, emailcontact, marketing, payment, product,
recommendation, shipping.
Respond only with the category name."
user : "checkoutordercommand,int32,checkout,info"
assistant : "order"
```

Listing 4: EShopOnContainers Checkout API Prompt

is selected if the response includes an enumeration of labels. Listing 4 shows the query employed to classify the EShopOnContainers OAD file, depicted in Listing 1. In this classification example, the `assistant`'s response is accurate.

3.4 Experiment settings

Datasets – In this study, we exploited two datasets used in previous works [23]. The first dataset, named *eshopping*, is an extension of the one used by Morais et al. [23]. It comprises 17 API documents describing four microservices-based systems from the e-shopping domain proposed by Assunçao et al.[3]: *EshopOnContainers*, *Eshop-netcore*, *Shopping Cart*, and *Socksshop*. These API documents were originally unlabelled but could be matched to functional classes identified by Mendonça et al. [20]. They extracted 14 features (*Frontend, User, Payment, Catalog, Cart, Order, Product, EmailContact, Shipping, Marketing, Recommendation, Audit, Currency, and Location*) from six e-shopping microservices architectures (cf. [3]). We used these features as the target labels in our controlled experiments.

The second dataset, *apisguru*, comprises data extracted from APIs.Guru [2], a repository of public API documentation containing 3826[4] API documents formalized using the OpenAPI standard, 2117 defined using OAS version 2.0, and 1709 defined using version 3.0. APIs.Guru allows adding a particular parameter to API documents, the *apisguru-categories* tag, allowing contributors to manually specify a tag related to the functionality or domain of the described API, such as *financial*, *media*, or *entertainment*. Although this parameter is not mandatory, we noted that 96 % (3658) of the API documents in this repository were associated with at least one tag.

We identified 31 categories that we used as labels to which the API documents should be classified. To enhance the automatic analysis of classification results, we excluded OADs having multiple tags (579 OADs). Furthermore, a random exploration of the dataset revealed significant imbalances among certain categories. For instance, the category *cloud* contained 2157 documents, accounting

[4] As of May 2024. This number excludes documents we were unable to open due to various parsing errors.

Table 1: Summary of the Experiment's Settings.

Datasets	KeyBERT	GPT
eshopping (17 OADs, 15 labels) *apisguru* (923 OADs, 31 labels)	`msmarco-distilbert-cos-v5`	`gpt-3.5-turbo` `gpt-4-turbo`

for 56 % of the entire dataset. Furthermore, the *cloud* label seemed more related to the API provider industry rather than to the actual functionality delivered by the API described in the OAD. Thus, we decided to exclude APIs tagged as *cloud* to maintain the reliability of our experiment. The final dataset consisted of 923 OADs. We relied on this dataset to evaluate the generalizability of our approach.

Tools – All experiments were performed on a Google Colaboratory [5] Notebook using the free plan with the following specifications: Intel Xeon CPU 2.20 GHz, Tesla T4 GPU, and 13GB of RAM. Datasets management was handled with Pandas [19], while NumPy [12] and Scikit-Learn [29] were used for metrics calculation. The extraction of relevant words was performed using KeyBERT and the pre-trained sentence transformer model `msmarco-distilbert-cos-v5` [14, 32]. The models used for classifying OADs were `gpt-3.5-turbo` [26] and `gpt-4-turbo` [27] for both the small and large datasets. Table 1 summarizes the experiment settings.

4 Results

In this section, we present the results of our experiments from the *eshopping* dataset, which consists of 17 OADs and 15 labels, as well as the applicability results from the *apisguru* dataset, which comprises 923 OADs and 31 labels. We tested each dataset with GPT–3.5 and GPT–4 in order to assess the model's respective capabilities in the same API classification task. To evaluate the performance of our classification method, we used macro precision ($Precision_M$), recall ($Recall_M$), and F1-score ($F1\text{–}Score_M$) metrics [36], ensuring that each class of our imbalanced datasets contributed equally to the overall results. All the compiled data can be found in Table 2. The findings provide valuable insights into the effectiveness of the proposed approach and shed light on the impact of various factors on API classification accuracy.

4.1 Overall Results

Our approach exhibited better performance than those obtained by Morais et al. [23] in a similar dataset (16 OADs); our *eshopping-dataset* was an extension of theirs. Indeed, our approach achieved a 100% F1-Score, whereas they obtained 71%.

Table 2: Results of the experiments.

Model	Dataset	$Precision_M$	$Recall_M$	$F1\text{–}Score_M$
`gpt-3.5-turbo`	*eshopping*	100.00	100.00	100.00
	apisguru	47.28	38.62	38.58
`gpt-4-turbo`	*eshopping*	100.00	100.00	100.00
	apisguru	47.75	40.71	39.12

Finding 1. The BERT-GPT API Classifier outperformed previous works by 29 points.

During prompt construction, we observed that the words extracted by KeyBERT comprised composed terms, such as *orderingapi*, *ordernumber*, and *orderitems*. We reproduced the experiment of Morais et al. [23] and found out that their approach could not handle such terms. Thus, we found that GPT was able to handle complex terms (not respecting syntactic rules) without data preprocessing. The same extends to acronyms. This finding could explain the precision improvement achieved by our approach.

Finding 2. The approach avoids the need for complex data preprocessing and is able to adapt to lexicon diversity and word structure in OADs.

Results also indicate that in our BERT-GPT classification method, the performance improvement from GPT–3.5 to GPT–4 is minimal, with both models achieving an F1-score of 100 on the *eshopping* dataset and approximately 39 on the *apisguru* dataset. Indeed, accounting for the variability and randomness in LLMs behaviour [8], the observed difference between the two versions of GPT is not significant enough to conclude that GPT–4 outperforms GPT–3.5.

Finding 3. Using GPT–4 negligibly improved the approach's performance.

4.2 Evaluation of Generalizability

We evaluated the generalizability of our approach using a more extensive and diverse dataset, the *apisguru* dataset, including 923 OADs from various sources and domains, providing an extended range of scenarios for evaluation. Indeed, the dataset's diversity, encompassing domains such as Finance, Education, Entertainment, and more, allowed us to examine how well the approach generalizes across different application domains. This enabled us to assess the approach's

performance in the face of variable data, providing valuable information on its practicality in real-world settings.

The experiment yielded key insights. We observed that the BERT-GPT classifier performance had substantially declined, dropping from an F1-score of 100 to around 39 for both GPT–3.5 and GPT–4. By manually analyzing 20 random labelling mismatches, we discovered that 18 of them were caused by inappropriate labels on the *apisguru* dataset. We also observed that GPT proposed additional labels not in the provided list, which, in most cases, were more accurate than those associated with OADs in the source dataset.

Table 3 shows a sample of the analyzed mismatches. In the first row, the *player*, *sportdata*, *mbl* and *nfldata* keywords point to the sports label, thus specifying the original entertainment label. In the second row, the *zipcode*, *area* and *getzipinfo* keywords clearly point to a location functionality, not a developer tool. In the third row, the *proxy* and *vnp* keywords point to a security feature, not a location one. The same is observed in the fourth and fifth rows where *charity*, and *marketing*, *advertise*, *ads* and *promotionsale* keywords point respectively to charity and marketing labels, not to e-commerce.

These observations suggest that the classifier's performance might not be indicative of its actual potential due to the dataset's labelling inaccuracies.

Table 3: Sample of analyzed mismatches.

Keywords	**Labels**		
	APIGuru	*GPT-3.5*	*GPT-4*
rotoballerarticlesbyplayer,rotoballerarticlesbyplayerid,playerinfo, rotoballer,**mlb**,**sportsdata**,**player**,players,**nfldata**,entries, rotoballerarticles,playerid,profile_image,apikeyheader,info	entertainment	sport_data	sport_data
zipcode,zip,interzoid,**getzipinfo**,providername,code,info,detailed, getzipcodeinfo,profile_image,developer_tools,www, areasquaremiles,city,information,**area**	developer_tools	location	location
ip2proxy,ip2location,**proxies**,proxy,profile_image,px2,px1,ip,px9, proxytype,**vpn**,px10,server,isproxy,providername,px11,lookup,ipv4	location	security	security
api_**charity**,**charity**_org,commerce_charity_v1_oas3,api_auth,array, api_scope,charity_org_id,charityorg,www,charityorgid,charityorgs, ebay_gb,**charity**,website,ebay_us,charitysearchresponse,org,com, providername,clientcredentials,ebay,charitable,support,link,users, marketplace,supported,application,profile_image,twitter,assistance, basepath,help,imageurl,search,user,servicename,implementation, page,html,access,errorid,10,subdomain,individual,accessing,server, ecommerce,developers,header,helps,associated	ecommerce	charity	charity
api_**marketing**,**marketing**,sell_marketing_v1_oas3,promote,**advertise**, promotes,promoting,ebay_it,item_promotion,ebay_au,promotions, promotional,**promotionsale**,**ads**,ebay,ebay_us,marketplaceid,selling, ebay_fr,ebay_de,ebay_gb,bulk_update_**ads**_bid_by_listing_id, promotion_name,promotiondetail,supportedmarketplaces,sales promotion_type,create_**ads**_by_inventory_reference,teasers,attract, bulk_create_**ads**_by_listing_id,promotionreportdetail,marketplace, bulk_update_**ads**_bid_by_inventory_reference,advertised, bulk_create_**ads**_by_inventory_reference,ebay_es,sells, listing_quantity_sold,**advertising**_eligibility,**promotion**, get_**ads**_by_inventory_reference,promoted,**promotion**type, bulk_update_**ads**_status_by_listing_id,**promotion**typeenum, ad_campaign,promotionid,active_seller_count,marketplace_id	ecommerce	marketing	marketing

Finding 4. GPT provided additional labels from those provided by the user, opening opportunities to handle APIs belonging to unknown classes or improving the precision in classifying APIs.

5 Discussion

The findings of our study hold several implications for API classification. First, our experiment shows the potential of the combined BERT-base and GPT-based classifier in accurately labelling API descriptions. The approach demonstrated high accuracy rates on a controlled dataset, and the evaluation of its generalizability offered promising prospects for its application to solve practical problems.

Second, the experiments demonstrated GPT's capacity to handle acronyms, abbreviations and composed terms without complex data preprocessing. Indeed, GPT directly processed the terms extracted from API descriptions, which allowed it to effortlessly adapt to the diverse terminologies and structures in the OADs.

Third, findings demonstrated the limited performance increase between GPT–4 and GPT–3.5 (+0.54 for the F1-Score). However, one must consider additional aspects when choosing between using one or the other, e.g., model utilization costs. GPT–4 was 20 times more expensive than GPT-3.5 at the time of our experiments. Therefore, the marginal performance gain of GPT–4 must be carefully considered in light of its increased costs.

Fourth, findings demonstrated the approach's capability to suggest alternative labels when judged more representative of the API intent, which dropped the approach accuracy on the *apisguru* dataset. Further research, particularly time-consuming manual evaluation, is required to dive deeper and provide conclusive answers to the extent to which the quality in the *apisguru* dataset causes the performance drop—which is what we currently can only hypothesize based on our random sample tests. Nevertheless, GPT suggestions of alternative labels open exciting research opportunities to explore applying the proposed approach to identifying unknown labels, detecting mislabelling, and improving the precision in existing API classification approaches.

Finally, the experiments were conducted with a limited sample of API documents, and their characteristics may only partially represent how OADs are created. We relied on specific ML pre-trained models, introducing limitations inherent to these models. Additionally, the effectiveness of the classification may be influenced by the prompt used to query the classifier. The lack of transparency in how GPT arrives at its decisions, also known as the "black-box" nature, makes it challenging to interpret the specific factors or features that heavily influence the classification outcomes [8, 22].

Consequently, our ability to validate and explain individual classifications may be restricted. Without a comprehensive understanding of GPT's decision-making process, we cannot ascertain the precise reasons behind its classifications. We could rely on prompt instructions to unveil the rationale behind each

labelling. However, it would require a manual, time-consuming analysis of each explanation, which is out of the scope of this exploratory study. This lack of explainability of GPT choices may introduce an element of uncertainty and limit the extent to which we can fully trust and interpret these experiments' results.

Despite employing a quantitative approach, the exploratory nature of our study allowed for some level of subjectivity in data interpretation. The use of automated algorithms and data processing techniques aimed to reduce bias; however, the potential for subjective decisions in data handling and analysis remains a limitation to consider when interpreting the results.

6 Future Plans

This exploratory study unveiled various challenges and perspectives toward applying LLM for API labelling using OpenAPI documents. A critical root challenge in this context is the creation of a larger curated dataset that allows researchers to evaluate the performance of LLM-based approaches in this task comprehensively. Therefore, we are building a process and tool to support the creation of such a dataset. Besides, using a proprietary LLM model imposes usage limitations induced by costs, and accessing these models through an API restricts the size of the prompts due to payload limitations. To cope with these challenges, we are currently exploring using open-source LLMs (e.g., LLAMA3, Gemma 2, Mistral, and Mixtral) deployed locally, working with the hypothesis that doing so could simplify the approach by avoiding the need to restrain the data sent to the LLM model, i.e., eliminating the use of KeyBERT to limit the amount of sent data. Similarly, we are collaborating with an industry partner to collect evidence of the effectiveness of such an approach in solving practical problems related to API mining and understanding.

7 Conclusion

This exploratory study suggested and experimented with a practical solution for efficient and accurate API classification based on state-of-the-art LLM models. The simplicity of data handling contributes to the approach's overall effectiveness. By bypassing the need for extensive data preprocessing, our approach achieved a remarkable enhancement in efficiency and simplicity in implementing an API classifier.

Eventually, we would like to stress that our approach pointed us to the inadequate or imprecise classification of openly available datasets. On the one hand, this is a severe threat to all research performed with these datasets; on the other hand, random sample manual investigation led us to the conclusion that the classification proposed by our approach could be more adequate and precise. We thus see a further use case of our approach in revising existing API classifications and also see a call to action for the community to clean and improve the available datasets.

Acknowledgments. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 06351.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Apis.Guru: Apis.guro apis repository. https://github.com/APIs-gurul (2021)
3. Assunção, W.K., Krüger, J., Mendonça, W.D.: Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In: Proceedings of the SPLC (A). pp. 22.1–22.6 (2020)
4. Balkus, S.V., Yan, D.: Improving short text classification with augmented data using gpt-3. Natural Language Engineering pp. 1–30 (2022)
5. Bisong, E., Bisong, E.: Google colaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners pp. 59–64 (2019)
6. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
7. Casas, S., Cruz, D., Vidal, G., Constanzo, M.: Uses and applications of the openapi/swagger specification: a systematic mapping of the literature. In: 2021 40th International Conference of the Chilean Computer Science Society (SCCC). pp. 1–8. IEEE (2021)
8. Chen, L., Zaharia, M., Zou, J.: How is chatgpt's behavior changing over time? arXiv preprint arXiv:2307.09009 (2023)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
10. González-Mora, C., Barros, C., Garrigós, I., Zubcoff, J., Lloret, E., Mazón, J.N.: Applying natural language processing techniques to generate open data web apis documentation. In: Web Engineering: 20th International Conference, ICWE 2020, Helsinki, Finland, June 9–12, 2020, Proceedings 20. pp. 416–432. Springer (2020)
11. Grootendorst, M.: Keybert: Minimal keyword extraction with bert. (2020). https://doi.org/10.5281/zenodo.4461265, https://doi.org/10.5281/zenodo.4461265
12. Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., et al.: Array programming with numpy. Nature **585**(7825), 357–362 (2020)
13. Hosono, M., Washizaki, H., Fukazawa, Y., Honda, K.: An empirical study on the reliability of the web api document. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). pp. 715–716. IEEE (2018)
14. HuggingFace: msmarco-distilbert-cos-v5. https://huggingface.co/sentence-transformers/msmarco-distilbert-cos-v5
15. Initiative, O.: Openapi specification v3.1.0. https://spec.openapis.org/oas/latest.html (2021)
16. Kalyan, K.S.: A survey of gpt-3 family large language models including chatgpt and gpt-4. Natural Language Processing Journal p. 100048 (2023)

17. Klein, T., Nabi, M.: Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds. arXiv e-prints pp. arXiv–1911 (2019)
18. Korde, V., Mahender, C.N.: Text classification and classifiers: A survey. International Journal of Artificial Intelligence & Applications **3**(2), 85 (2012)
19. McKinney, W., et al.: pandas: a foundational python library for data analysis and statistics. Python for high performance and scientific computing **14**(9), 1–9 (2011)
20. Mendonça, W.D., Assunção, W.K., Estanislau, L.V., Vergilio, S.R., Garcia, A.: Towards a microservices-based product line with multi-objective evolutionary algorithms. In: 2020 IEEE Congress on Evolutionary Computation. pp. 1–8 (2020)
21. Meng, M., Steinhardt, S., Schubert, A.: Application programming interface documentation: What do software developers want? Journal of Technical Writing and Communication **48**(3), 295–330 (2018)
22. Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J.: Deep learning–based text classification: A comprehensive review. ACM Comput. Surv. **54**(3) (apr 2021). https://doi.org/10.1145/3439726, https://doi.org/10.1145/3439726
23. Morais, G., Adda, M., Hadder, H., Bork, D.: x 2omsac-an ontology population framework for the ontology of microservices architecture concepts. In: World Conference on Information Systems and Technologies. pp. 263–274. Springer (2023)
24. Morais, G., Lemelin, E., Bork, D., Adda, M.: Companion source code repository. https://github.com/UQAR-TUW/enhancing-api-labelling-bert-gpt (2024)
25. Norvig, P., Russell, S.: Artificial intelligence: a modern approach. Pearson, Harlow **1**, 1239–1269 (2021)
26. OpenAI: Gpt-3.5-turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo
27. OpenAI: Gpt-4-turbo. https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4
28. OpenAI: openapi-python. https://github.com/openai/openai-python
29. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
30. Alexandre Peixoto de Queirós, R., Simões, A., Pinto, M.: Code Generation, Analysis Tools, and Testing for Quality. Advances in Computer and Electrical Engineering (2327-039X), IGI Global (2019), https://books.google.ca/books?id=Ieh_DwAAQBAJ
31. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
32. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019), http://arxiv.org/abs/1908.10084
33. da Rocha Araujo, L., Rodríguez, G., Vidal, S., Marcos, C., dos Santos, R.P.: Empirical analysis on openapi topic exploration and discovery to support the developer community. Computing and Informatics **40**(6), 1345–1369 (2021)
34. Sanders, R.: The pareto principle: its use and abuse. Journal of Services Marketing **1**(2), 37–40 (1987)
35. Serbout, S., Pautasso, C., Zdun, U., Zimmermann, O.: From openapi fragments to api pattern primitives and design smells. In: 26th European Conference on Pattern Languages of Programs. pp. 1–35 (2021)
36. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Information processing & management **45**(4), 427–437 (2009)

37. Uddin, G., Robillard, M.P.: How api documentation fails. Ieee software **32**(4), 68–75 (2015)
38. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)