

# Is It Professional or Exploratory? Classifying Repositories Through README Analysis

Maximilian Auch<sup>1</sup>, Maximilian Balluff<sup>1</sup>, Peter Mandl<sup>1</sup> and Christian Wolff<sup>2</sup>

<sup>1</sup>IAMLIS, Munich University of Applied Sciences HM, Lothstraße 34, 80335 Munich, Germany

<sup>2</sup>I:IMSK, University of Regensburg, Universitätsstraße 31, 93040 Regensburg, Germany

**Keywords:** Classification, LLM, README, Zero-Shot, Few-Shot.

**Abstract:** This study introduces a new approach to determine whether GitHub repositories are professional or exploratory by analyzing README.md files. We crawled and manually labeled a dataset that contains over 200 repositories to evaluate various classification methods. We compared state-of-the-art Large Language Models (LLM) against traditional Natural Language Processing (NLP) techniques, including term frequency similarity and word embedding-based nearest-neighbors, using RoBERTa. The results demonstrate the advantages of LLMs on the given classification task. When applying a zero-shot classification without multi-step reasoning, GPT-4o had the overall highest accuracy. The implementation of a few-shot learning showed a mixed result in different models. Llama 3 (70b) achieved 89.5% accuracy when using multi-step reasoning, though such improvements were not consistent across all models. Also, our experiments with word probability threshold filtering showed mixed results. Our findings highlight important considerations regarding the balance between accuracy, processing speed, and operational costs. For time-critical applications, we found that direct prompts without multi-step reasoning provide the most efficient approach, while the model size made a smaller contribution. Overall, README.md content proved sufficient for accurate classification in approximately 70% of cases.


## 1 INTRODUCTION


Determining the nature and maturity of software projects is an important task for a variety of reasons. Relying on software that is not maintained professionally can cause security vulnerabilities, bugs, and other problems. Even the extraction of knowledge from software repositories, like learning from source code, integration, or the decisions made, can depend on the quality of the repository. In a current research project, we investigate these possibilities for extracting technology decisions using ML. It is particularly important that the database is reliable when it comes to the automated generation of recommendations for use or the decision to migrate between technologies. This is why we present an automated classification attempt to determine whether a software repository is professionally maintained or if it is for exploration purposes by using the README.md files. These markdown


files are often available and presented in repositories on hosting platforms, like GitHub and are typically used to describe the project, its purpose, and how to use it. In doing so, in this paper the following novel research question is addressed:


Can we determine whether a software repository is professionally maintained or if it is for exploration purposes using the README.md files?

Furthermore, in addition to the presentation of a new annotated dataset and an evaluation to clarify the research question, the work includes a discussion of the methodologies and results, considering factors such as classification time and costs. The structure of this paper is as follows: Section 2 covers related work, followed by the definition of professional and exploratory projects in Section 3 and details about the dataset created in Section 4. The methodology for classification is addressed in Section 5. Section 6 presents the results, while Section 7 provides a discussion of these findings and key observations, followed by Section 8, which shows limitations. Finally, Section 9 concludes the paper and Section 10 gives an

<sup>a</sup> <https://orcid.org/0000-0002-4860-7464>

<sup>b</sup> <https://orcid.org/0000-0002-6837-0628>

<sup>c</sup> <https://orcid.org/0000-0003-4508-7667>

<sup>d</sup> <https://orcid.org/0000-0001-7278-8595>

overview for future work.

## 2 RELATED WORK

Researchers have investigated various approaches to classify software repositories, drawing insights from documentation like README.md files and other metadata sources. Though existing literature has not specifically labeled repositories as "professional" or "exploratory," scholars have examined README.md content to establish different classification schemes.

Prana et al. (Prana et al., 2019) examined README.md files from 393 GitHub projects by manually annotating Sections in documents. They built a classification system to identify common content types, including "What" and "How" categories. Their system reaches an F1-score of 0.75, confirming README.md files as rich sources of repository insights. The team's findings revealed a crucial gap: many files lacked sufficient information about project purpose and status, pointing toward possibilities for more sophisticated classification methods.

Petrovic et al. (Petrovic et al., 2016) tackled the challenge of finding similar GitHub projects by combining textual descriptions with user engagement metrics, including forks and stars. Their solution featured an Artificial Neural Network (ANN) with an auto-encoder to reduce dimensions, with similarity assessments based on euclidean distance calculations. Despite not addressing the classification into professional and exploratory project, their work reinforces the value of README documentation in repository classification efforts.

Xu et al. (Xu et al., 2017) developed REPERSP, a recommendation system for discovering software projects. Using TF-IDF and cosine similarity on both documentation and source code, they incorporated user behavior (e.g., forks and stars) for a more comprehensive similarity measure. While aimed at recommendations, this approach could extend to distinguishing repository types.

Zhang et al. (Zhang et al., 2017) created RepoPal to detect similar GitHub repositories by analyzing README.md files and user-interaction data, employing TF-IDF vectors and cosine similarity. Their methodology, which focused on project similarity, could potentially classify repositories by maintenance level.

Capiluppi et al. (Capiluppi et al., 2020) used a graph-based model to analyze semantic relationships in Java projects by examining the README.md and pom.xml files. Their work illustrates that detailed documentation analysis reveals key repository char-

acteristics, relevant for professional-exploratory classification.

Rao and Chimalakonda (Rao and Chimalakonda, 2022) explored artifact similarity in Java projects, transforming text (e.g., pull requests, issues, commit messages, README.md) into vectors with Sentence-BERT and measuring similarity through cosine similarity. Though their focus was finding similar repositories, their results highlight README.md files as critical for various classification tasks.

## 3 PROFESSIONAL OR EXPLORATORY PROJECTS

First it is necessary to define what exactly is meant by professional and exploratory projects in context of this work. Since there is no general definition of these terms, we will provide a definition, which was conducted inductively based on the crawled README.md files. The following criteria are used to determine whether the software in a repository is professionally maintained or rather has a similar purpose or if it rather has some kind of exploration purpose:

### Professional Projects Contain...

1. **Open-Source Projects:** Software projects that are publicly available and developed collaboratively by a community of contributors.
2. **Internal Corporate Projects:** Software projects developed within an organization to support its internal operations, processes, and workflows.
3. **Commercial/Revenue-Generating Projects:** Software projects that are developed and sold as commercial products or services to generate revenue for the organization.
4. **Commissioned/Contract-Based Projects:** Software projects undertaken by a development team or agency on behalf of a client, with a specific set of requirements and a defined scope.
5. **Enterprise-Grade Projects:** Software projects that are designed and developed to meet the stringent requirements of large-scale, mission-critical enterprise systems.
6. **Production-Ready Components:** Software projects developed to provide reusable components, modules, libraries, or widgets that can be integrated into other software systems for production-ready applications.

## Exploratory Projects Contain...

1. **Tutorials/Examples:** Software projects that are primarily created for educational or demonstration purposes to teach specific programming concepts, techniques, or best practices. These projects are often small in scope, focused on a particular feature or functionality, and intended to be easily understood and replicated by learners.
2. **Prototypes:** Software projects that are built to validate an idea, test the feasibility of a concept or explore potential solutions to a problem. Prototypes are typically not intended for production use, but rather to gather feedback, identify technical challenges, and inform the development of a more polished, production-ready application.
3. **Proofs of Concept (PoC):** Similar to prototypes, PoCs are developed to demonstrate the viability of a particular approach, technology, or solution to a problem. These projects are often small in scale and focused on validating a specific hypothesis or claim rather than building a complete, production-ready application.
4. **School/Training Projects:** Software projects created as part of educational programs, such as university courses, coding bootcamps, or internal training initiatives. These projects focus primarily on the learning process, allowing students to apply their knowledge and develop practical programming skills.
5. **Research Projects:** Software projects that are part of research efforts, focused on showcasing novel technologies, algorithms, or techniques without emphasizing long-term aspects such as software architecture or intending to make the system productive.
6. **Technology Exploration/Testing:** Software projects developed to experiment with new technologies, languages, frameworks, or tools, without a specific production-oriented goal in mind. These projects are often small-scale, exploratory in nature and may not have a clear end-user or business value.
7. **Experimental/R&D Projects:** Software projects that are part of research and development efforts, focused on advancing the state-of-the-art in a particular domain or exploring innovative solutions to complex problems. These projects may not have immediate commercial or practical applications but are driven by a desire to push the boundaries of what is possible with technology.
8. **Hobby/Personal Projects:** Software projects developed by individuals for their own personal in-

terest, enjoyment, or learning, without a focus on commercial or enterprise-level requirements. These projects are often driven by the creator's passion, curiosity, or desire to explore a particular domain or technology.

## 4 DATASET

To apply different techniques for classification, we needed a dataset. Since we did not find related work or data, which we can use, we crawled, annotated and analyzed README.md files from software project repositories on GitHub. In the following we describe the data collection and annotation process, as well as giving some insights into this new dataset, which is publicly available at:

<https://github.com/CCWI/sw-repo-classification-study>

In addition to the data, the linked repository provides the classification source code, including the prompts and instructed output structure for each experiment.

### 4.1 Data Collection

Since no existing dataset classifies software repositories in a comparable way, we curated and annotated our own. Using the GitHub API, we extracted README.md files from various repositories, focusing solely on this common format and excluding less prevalent formats like README.rst, which are beyond our study's scope. The final dataset comprises 200 randomly selected repositories from GitHub, each containing at least one README.md file.

In general, repositories are very heterogeneous in terms of structure and content. They can contain a wide variety of files. One of the most important files in a repository is the README.md file, which typically provides an overview of the project, its purpose, how to use it, and other relevant information. It is often present in the root directory of the repository and is displayed automatically on repository's hosting platforms, like GitHub and GitLab pages. The README.md file is written in markdown format. In some cases, multiple README files may be present in different directories of the repository to provide information about standalone or additional software, as well as software components, which are typically included in the main software as submodules.

Another aspect to consider is time. README files can be changed and updated several times during the development of a project. This can result in

a README file of a project that is in an early stage of development containing less detailed information than the README file of a project that is in an advanced stage of development. To take these differences into account, we also crawled all available versions of a README file and removed duplicates.

By analyzing the crawled files to identify a common structure, we found that README.md files provide a comprehensive overview of a project, guiding users from an understanding of the project's nature and purpose to an appreciation of how they can utilize and contribute to it. Typically, some of the following categories of information, which were found and summarized by Prana et al. (Prana et al., 2019) and from our dataset, are provided:

1. **What:** A title and an introduction or overview of the project, usually at the beginning. Often, at least a title is available.
2. **Why:** Comparisons to other projects or advantages of the current project. (Not common)
3. **How:** The most frequent category, including instructions on usage, configuration, installation, dependencies, and troubleshooting.
4. **When:** Information about the status, versions, and functionality of the project (complete or in progress).
5. **Who:** Credits, acknowledgments, license information, contact details, and code of conduct.
6. **References:** Links to additional documentation, support resources, translations, and related projects.
7. **Contribution:** Instructions on how to contribute to the project, including forking or cloning the repository.

## 4.2 Annotation

After crawling, 200 repositories were randomly selected for manual annotation. The annotation was carried out by two people who read the README.md files of the repositories and categorized the projects as 'professional', 'exploratory' and 'not assignable' according to their assessment. The third label 'not assignable' was introduced because in some cases the README files did not contain enough information to make a clear classification. Prana et al. (Prana et al., 2019) already mentioned that many README files lack information regarding the purpose and status of the project. We found that often README files do not contain any information, in addition to a header and a few words. When the header and the description

body are not clear enough to make a classification, the project was labeled as 'not assignable'.

The annotations were then compared, and all discrepancies were discussed in order to reach a full agreement. The definition was further refined to better separate the classes for cases that are difficult to categorize. A total of 200 annotated repositories were used as test data. A negligible amount of data was used to create and optimize LLM prompts and to validate the classical NLP techniques. Text in languages other than English was translated using Google Translate and DeepL.com to ensure that the annotators could understand the content of the README files.

As a result, we created a test dataset with 74 exploratory, 65 professional, and 61 not assignable repositories.

## 4.3 Data Insights

To get an idea about the annotated README texts, we checked various features and patterns that distinguish professional software projects from exploratory projects and collected meta-data on software repositories. We analyzed the subject areas (topics), given languages and qualities of the repositories. In Figure 1 the text length of all texts and by each label is provided. It is noticeable that exploratory labeled text is smaller with a mean of 998 characters, while READMEs of repositories with label professional has a mean of 3374 characters. However, we have noticed that there are repositories that are still in the initial phase and only provide a small README text, but the description indicates a professional background. For example, the following README.md text is from a repository with currently 23 stars, 16 watches, 8 forks and around 800 commits, which is versioned in a commit from 2016 in the Repository <https://github.com/CMSgov/bluebutton-data-server> (accessed on 27th September 2024):

### "CMS Blue Button Server

The CMS Blue Button project provides Medicare beneficiaries with access to their health care data and supports an ecosystem of third-party applications that can leverage that data. This project provides the FHIR server used as part of Blue Button."

By analyzing the languages used in the README files, we found that the majority of the README files are written in English. Besides that, we found the README.md files to cover a wide range of project types, including web applications, libraries and frameworks, data analysis tools, DevOps and

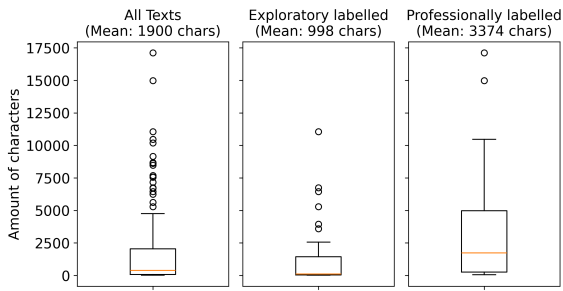


Figure 1: Distribution of text lengths in the README files.

deployment tools, microservices, IoT and hardware-related projects. While Java is the most frequently mentioned programming language, other languages such as Python, JavaScript, and C# are included as well.

## 5 METHODOLOGY

Classifying software components as exploratory or professional requires understanding the context of the README texts. Traditional machine learning methods need extensive training data for reliable models, but advances in large language models (LLMs) now enable effective text classification with minimal training data. Given their ability to handle extensive text and contextual information, LLMs are promising for this task.

The core approach transforms text into word embeddings, representing them as vectors in an  $n$ -dimensional space for classification. Birunda and Devi (Selva Birunda and Kanniga Devi, 2021) reviewed embedding methods, categorizing them as: traditional, static, and contextualized. For our dataset, we tested models from all categories, emphasizing contextualized embeddings, including smaller pre-trained models like BERT and RoBERTa, and modern LLMs.

We explored optimization strategies for LLMs, such as zero-shot, few-shot learning, multi-step reasoning, and word probability filtering. Techniques like Retrieval Augmented Generation (RAG) (Gao et al., 2024) and fine-tuning were not feasible due to the limited size of the dataset. Similarly, ensemble methods were excluded to maintain cost and time efficiency.

The following Section details the models and approaches we applied:

### 5.1 TF-IDF and Similarity Measures

At first, we tried a zero-shot text classification method leveraging Term Frequency-Inverse Document Frequency (TF-IDF) combined with a cosine similarity which was also applied by related work, like Xu et al. (Xu et al., 2017) or Zhang et al. (Zhang et al., 2017). This method is categorized by Birunda and Devi (Selva Birunda and Kanniga Devi, 2021) as "traditional word embeddings".

First we define categories using the definition of each class from Section 3. Then we convert all input texts into high-dimensional vectors that represent the importance of words within the context of the corpus using TF-IDF. Subsequently, we calculate the cosine similarity between the vector representation of the input text and those of the predefined category descriptions. The category that yields the highest similarity score to the input text is selected as the classification outcome.

To improve the robustness of the classification and prevent misclassifications due to low similarity scores, we implemented a confidence-based rejection mechanism. This mechanism allowed the classifier to abstain from making uncertain predictions by labeling inputs with low similarity as "not assignable." For this we find for each input  $x$  the class  $c$  with the highest cosine similarity score  $\text{sim}(x, c)$  among the classes  $c \in C$  and apply the threshold  $t$ . The classification rule can be described as:

$$\hat{y}(x) = \begin{cases} \arg \max_{c \in C} \text{sim}(x, c) & \text{if } \max_{c \in C} \text{sim}(x, c) \geq t \\ \text{"not assignable"} & \text{if } \max_{c \in C} \text{sim}(x, c) < t \end{cases}$$

Adjusting  $t$ , we control the confidence level of the classifier. A higher threshold resulted in fewer but more confident classifications, while a lower  $t$  increased coverage but included less certain predictions. Then we optimized the threshold against the applied evaluation metrics.

### 5.2 Nearest-Neighbor Search

We further explored a classification approach using word embeddings combined with a nearest-neighbor algorithm from the category "static word embeddings". In this method, both the input texts and the category descriptions are represented as vectors by averaging the embeddings of their constituent words. This approach captures the semantic information of the texts and categories within a continuous vector space.

The classification process involves the following steps:

1. **Compute Embeddings:** Calculate the embedding for each class description and each input text by averaging the word embeddings from a pre-trained word embedding model.
2. **Build Nearest-Neighbor Model:** Use the embeddings of the class descriptions to create a nearest-neighbor model based on cosine distance.
3. **Classify Input Texts:** For each input text, find the nearest class by identifying the class description whose embedding is closest to that of the input text.
4. **Assign Labels:** Assign the input text to the nearest class, except the probability lies under a distance threshold  $t_d$ , in which case a confidence-based rejection mechanism, like in the approach from 5.1.

Formally, the embeddings for each class description  $c \in C$  are calculated and then a nearest-neighbor model with cosine distance is used to find the closest class for each input text  $x$ . While  $\text{dist}(x, c)$  is the cosine distance between embeddings, the classification rule was implemented in the following way:

$$\hat{y}(x) = \begin{cases} \arg \min_{c \in C} \text{dist}(x, c), & \text{if } \text{dist}(x, c) \leq t_d, \\ \text{"not assignable"}, & \text{if } \text{dist}(x, c) > t_d. \end{cases}$$

We optimize  $t_d$  against accuracy and F1-score to maximize performance metrics.

### 5.3 Transformer-Based Language Model

In some cases, the README file may contain information relevant to the classification task, but it can be buried in a large amount of text or is not clearly labeled. This can make it challenging to extract the relevant information and use it for classification. This is a task where transformer models with an attention mechanism (Vaswani et al., 2017) can be particularly useful, as they are context aware. This is why the review (Selva Birunda and Kanniga Devi, 2021) categorize them as "contextualized word embeddings".

For this study we chose a large RoBERTa model from FacebookAI (Liu et al., 2019), which is a fine-tuned version of BERT for the zero-shot classification task.

### 5.4 Large Language Models (LLM)

Current LLMs are typically transformer-based language models, like BERT or RoBERTa, but are trained on a larger amount of data and come with more parameters. This is why the benefits described

in Section 5.3, such as context awareness, also apply to LLMs. LLMs can better understand both the classification task context and the input text context, extracting relevant information even when it is implicit or hidden in large text. With their larger input (context) size, they support advanced techniques like reasoning, leading to improved performance on zero-shot classification tasks.

In this study, we used several LLMs to classify the software repositories and some advanced techniques to optimize the results. In the following, the selected models and the applied techniques are described.

#### 5.4.1 Selected Models

We chose comparable models from different providers, including OpenAI, Anthropic, Google, and Meta. We selected the most popular and largest models, as well as smaller versions of these models for comparison. The selected models are the following.

1. OpenAI
  - **Large:** GPT-4 (Version: 08-06)
  - **Small:** GPT-4-Mini (Version: 07-18)
2. Anthropic
  - **Large:** Claude 3.5 Sonnet (Version: 06-20)
  - **Small:** Claude 3 Haiku (Version: 03-07)
3. Google
  - **Large:** Gemini 1.5 Pro (Version: 001)
  - **Small:** Gemini 1.5 Flash (Version: 001)
4. Meta
  - **Large:** Llama 3.1 (405B parameters)
  - **Medium:** Llama 3 (70B parameters)
  - **Small:** Llama 3 (8B parameters)

Similar prompts and configurations were used for all models, including a temperature setting of 0.0, to ensure the most consistent results possible.

#### 5.4.2 Multi-Step Reasoning with Structured JSON Output

Various prompting techniques have shown that LLMs can perform complex reasoning tasks without task-specific few-shot examples. Kojima et al. (Kojima et al., 2022) introduced zero-shot Chain of Thought (Zero-shot-CoT), a method that elicits step-by-step reasoning from LLMs using a simple prompt like adding "Let's think step by step". Building upon this concept, we apply a modified approach that combines multi-step reasoning with structured JSON output. This method incorporates reasoning elements directly into the JSON structure. For the multi-step-reasoning,

we instruct the LLM to discuss, pre-classify the text for each subclass, verify the chosen subclasses and summarize the reasoning by comparing all arguments, before a final classification should be performed. We also request to classify each class separately first and then combined, to get insights into each class probability. The approach using multi-step reasoning should ensure that the model has gone through a comprehensive reasoning process before arriving at its conclusion. By explicitly prompting the model for intermediate steps and explanations, we encourage it to engage in more detailed and transparent reasoning.

We apply the procedure to all selected LLMs to test whether possible improvements in the results of the respective models. For multi-step reasoning, we tested different output instructions with the various LLMs.

### 5.4.3 Word Probability Filtering

Similar to the confidence-based rejection mechanism, we implemented in TF-IDF and cosine similarity approach as well as the nearest-neighbor search, we are able to perform a probability-based relabeling if the LLM / API provides the probabilities of the generation tokens.

In case of OpenAI, the API provides such logarithmic probabilities, which we used to optimize a threshold against accuracy and F1-score. In this study we compared the results for all models which support this and discuss them in the following.

## 6 RESULTS

This Section presents the classification results, comparing all approaches using various performance metrics, including accuracy, precision, recall, F1-score, and MCC. Table 1 details the outcomes of the zero-shot classification approach, categorized by different methodologies such as multi-step reasoning and word probability filtering. The best model and achieved metric of each category is marked bold. Additionally, the best result overall is underlined.

After evaluating the zero-shot approaches, the tests were repeated using the few-shot approach. Table 2 highlights the models that showed improvement in any performance metric. As before, the best-performing model and its corresponding metric are emphasized in bold.

## 7 DISCUSSION

The results show that most LLMs outperform the Non-LLM approaches in terms of accuracy, precision, recall, F1-score and MCC. Only the smallest Llama 3 (8b) model performed worse than a TF-IDF with cosine similarity in every test case. Interestingly, the large models Llama 3.1 (405b) and Gemini 1.5 Pro also did worse than the classical approach when multi-step reasoning is used. The overall best zero-shot results were achieved using current OpenAI's large model GPT4o, applying a direct prompt without any reasoning before classification. Among the smaller models, which are reasonably cheaper and faster in inference, Google's Gemini 1.5 Flash achieved the best results using direct prompts to perform a zero-shot classification. We used a few-shot approach, giving the LLMs several examples for each class as context before they made classifications. The results were mixed. Since the extension of a prompt introducing examples increases the costs and inference-time, we are only focusing on the LLMs, which improved from this approach. Compared to zero-shot results, we were able to increase the performance of Llama 3 (70b)<sup>f</sup> including multi-step reasoning to a point that it outperformed all zero-shot approaches and achieved the overall best result for our classification task with an accuracy of 89.5%. Also, Claude 3.5 Sonnet<sup>f</sup> improved by 4.5% to 88% accuracy. Both results were achieved by using multi-step reasoning.

The application of multi-level reasoning to improve the classification performance of LLMs overall led to inconsistent results. While certain models, including GPT4o-Mini, Claude 3.5 Sonnet and Llama 3 (70b), performed better with this approach, others showed a drop in performance. Figure 2 illustrates these different effects.

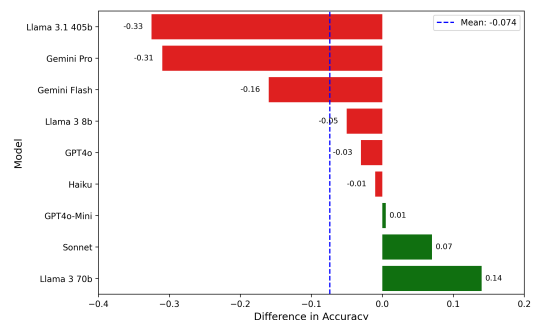


Figure 2: Changes in model performance by replacing a direct prompt with multi-step reasoning prompt.

On closer examination of the results, it is also noticeable that in our tests the use of multi-step reason-

Table 1: Comparison of zero-shot approaches, including multi-step reasoning and word probability filtering results.

API-Provider	Approach / Model	Accuracy	F1-score	Precision	Recall	MCC
Non-LLM Approaches						
Self-hosted	TF-IDF & Similarity	<b>0.615</b>	<b>0.623</b>	<b>0.629</b>	<b>0.621</b>	0.423
	Word2Vec & NN	0.590	0.539	0.647	0.616	<b>0.467</b>
	RoBERTa large	0.390	0.312	0.268	0.376	0.0681
LLMs (Direct Prompt)						
OpenAI	<b>GPT4o<sup>d</sup></b>	<b>0.890</b>	<b>0.890</b>	<b>0.893</b>	<b>0.889</b>	<b>0.835</b>
	GPT4o-Mini <sup>d</sup>	0.825	0.825	0.851	0.821	0.743
Google	Gemini 1.5 Pro <sup>d</sup>	0.820	0.816	0.845	0.813	0.736
	Gemini 1.5 Flash <sup>d</sup>	0.860	0.859	0.861	0.861	0.792
Anthropic	Claude 3.5 Sonnet <sup>d</sup>	0.765	0.741	0.820	0.753	0.667
	Claude 3 Haiku <sup>d</sup>	0.685	0.691	0.698	0.688	0.526
Self-hosted	Llama 3 (405b) <sup>d</sup>	0.810	0.794	0.849	0.799	0.730
	Llama 3 (70b) <sup>d</sup>	0.705	0.626	0.795	0.683	0.596
	Llama 3 (8b) <sup>d</sup>	0.575	0.564	0.643	0.564	0.374
LLMs (Multi-Step Reasoning Prompt)						
OpenAI	GPT4o <sup>r</sup>	<b>0.860</b>	<b>0.860</b>	<b>0.876</b>	<b>0.863</b>	<b>0.798</b>
	GPT4o-Mini <sup>r</sup>	0.830	0.830	0.831	0.830	0.745
Google	Gemini 1.5 Pro <sup>r</sup>	0.510	0.495	0.678	0.526	0.355
	Gemini 1.5 Flash <sup>r</sup>	0.700	0.697	0.711	0.696	0.552
Anthropic	Claude 3.5 Sonnet <sup>r</sup>	0.835	0.830	0.860	0.826	0.760
	Claude 3 Haiku <sup>r</sup>	0.675	0.667	0.707	0.690	0.540
Self-hosted	Llama 3 (405b) <sup>r</sup>	0.485	0.475	0.639	0.499	0.304
	Llama 3 (70b) <sup>r</sup>	0.845	0.836	0.858	0.838	0.774
	Llama 3 (8b) <sup>r</sup>	0.525	0.496	0.614	0.506	0.297
LLMs (Word Probability Filtering)						
OpenAI	GPT4o-Mini <sup>s</sup> <sub>Proba</sub>	<b>0.862</b>	<b>0.862</b>	<b>0.861</b>	<b>0.860</b>	<b>0.790</b>

Table 2: Comparison of the few-shot approach, which improved the LLM results.

API-Provider	Approach / Model	Accuracy	F1-score	Precision	Recall	MCC
Anthropic	Claude 3.5 Sonnet <sup>r</sup>	0.880	0.880	0.892	0.876	0.822
	Claude 3 Haiku <sup>d</sup>	0.700	0.708	0.714	0.704	0.548
Self-hosted	Llama 3 (70b) <sup>d</sup>	0.875	0.873	0.893	0.870	0.817
	<b>Llama 3 (70b)<sup>r</sup></b>	<b>0.895</b>	<b>0.893</b>	<b>0.896</b>	<b>0.893</b>	<b>0.843</b>
	Llama 3 (8b) <sup>d</sup>	0.595	0.559	0.745	0.600	0.459
	Llama 3 (8b) <sup>r</sup>	0.570	0.568	0.677	0.581	0.411

ing often led to LLMs categorizing fewer texts and instead rating them as "not assignable". As an example, we refer to a comparison of confusion matrices from the direct prompt and the multi-step reasoning prompt of GPT4o, Gemini 1.5 Pro and Claude 3.5 Sonnet in Figures 6, 7, 8. The results appear to be classified more cautiously or weighed up more carefully if the decision is discussed in detail beforehand and arguments for and against the respective class are explained. We observed that cases which aren't clearly defined often aren't assigned to either class, even though the instructions specify making a classification if there are any indications.

In summary, we are unable to say that models generally perform better in this particular use case classifying Repositories by README.md files using multi-step reasoning. However, especially in the case of the performance gains observed with Llama 3 (70b),

it should be noted that the open-weights model performs comparatively well against the large LLM models from Google, OpenAI and Anthropic.

Finally, we tested whether the results of the models could be improved by extracting the token probability for the respective label and changing it to 'Not assignable' if it fell below a certain threshold. Although in the case of GPT4o it did not provide any advantages and GPT4o-Mini<sup>r</sup> could hardly benefit from it, the simple approach GPT4o-Mini<sup>s</sup> shows significantly better results. The chart in Figure 3 visualizes how the accuracy of a model changes with different probability thresholds. The best accuracy of 86.2% was determined at a threshold of 91%. All classifications below this probability were re-labeled as 'not assignable', leading to improved results.

Since costs and speed are also an important factor for use on potentially millions of software repository



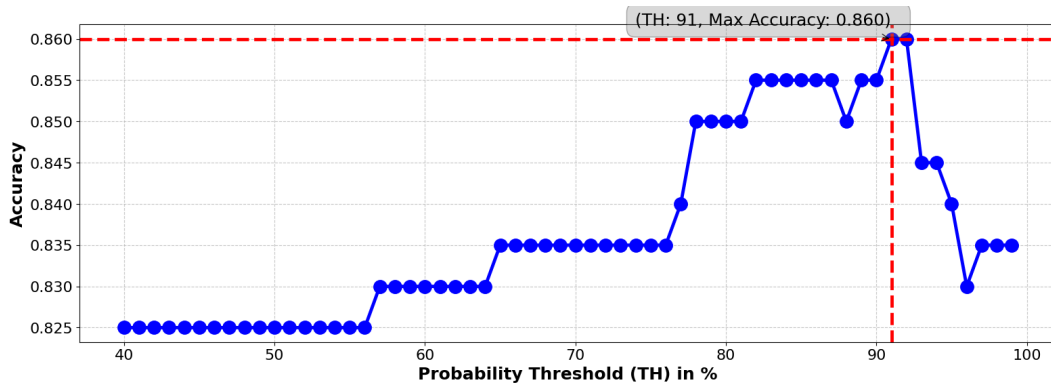


Figure 3: Threshold-Accuracy Curve for GPT4o-Mini<sup>s</sup>.

ries from GitHub and other hosting platforms, in addition to the measured metrics, we analyzed them as well. Since we did not ensure to have the best possible, comparable hardware or test settings, we would like to point out fundamental differences and emphasize their magnitude instead.

Figure 4 shows the differences between the most promising approaches by visualizing the average query time per README in seconds compared to the other LLMs. The measurements show that the direct query with the smallest input prompt and the smallest output is the fastest across all model sizes considered, while querying with reasoning during the inference led to significantly higher query times. The measurements also show that smaller LLM sizes result in faster queries, although the speed increase is much smaller compared to the requested output size.

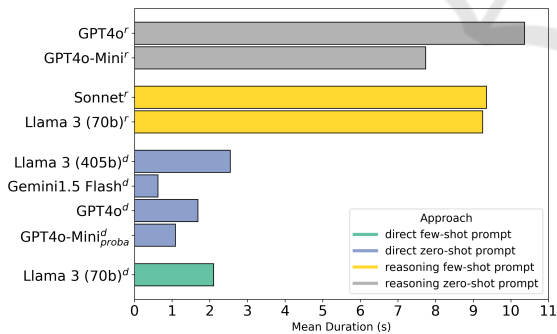


Figure 4: Mean duration of inference to classify the test data by each LLM in seconds.

While the request times favor the choice of a direct prompt without reasoning, making LLMs such as GPT4o<sup>d</sup> a good choice due to their good accuracy, the costs must also be considered as a further decision dimension. While we cannot determine the costs for a self-hosted LLM, such as Llama 3 or 3.1, we were at least able to use the current API costs of the providers of the other models for calculation. Figure 5 com-

pare the average costs of the approaches and models for classifying the test dataset in \$.

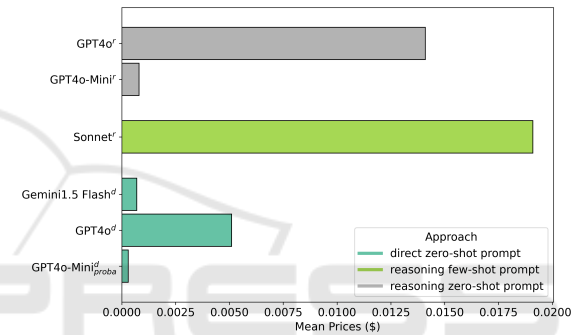


Figure 5: Mean prices of each inference to classify the test data by each LLM in \$.

The plot shows that the costs of the models depend primarily on the size of the model. Reasoning, which leads to more input and output tokens, increases the cost of a query as well, but less when for example comparing the costs of GPT4o-Mini<sup>d</sup> and GPT4o-Mini<sup>f</sup>.

In conclusion, it can be stated that a classification of software repositories is possible based on the README files, for around 70% providing enough information to label them as exploratory or professional, with LLMs delivering the best results. This is a sufficient start but should be increased in further work with additional features such as stars, watches, number of contributors, number of commits or other aspects such as licensing, if available, so that more projects can be clearly classified.

## 8 THREATS AND LIMITATIONS

The fast-growing number and rapid updates of LLM in combination with the various optimization approaches makes evaluation difficult. It is often only

possible to make use case and dataset-specific statements at a specific point in time instead of deriving generally applicable rules. We were unable to perform any model-specific prompt optimizations within the scope of the study due to the time and effort involved, which might have improved individual model results, such as the large Llama 3.2, which performed comparatively poorly considering its size.

It cannot be ruled out that in difficult cases the annotation of the README files was labeled incorrectly by both annotators. However, the annotations were discussed to reach an agreement. A larger number of annotators might reduce the possible error rate. Also consulting the authors of the repositories could lead to more accurate results but is time consuming and not feasible for a larger number of repositories.

Another problem could be the number of annotated repositories, which is relatively low. This could affect the validity of the results. A larger number of annotated repositories could lead to more accurate results. The selection of repositories for annotation was random, which could affect the representativeness of the results. A targeted selection of repositories that are representative of the different, domain-specific or technical categories or certain metadata, such as the number of stars or forks, could lead to more accurate results.

The clear separation of software repositories into the categories ‘professional’ and ‘exploratory’ is not always unambiguous. In some cases, it can be difficult to clearly categorize repositories. For example, there are professionally maintained tutorial repositories that can be classified as professional in certain contexts. In the context of our motivation behind the approach of this study to derive design decisions from these repositories, such cases fall into the exploratory category. Our intention is primarily to separate such tutorials from possibly productively used software to consider migrations between design elements, such as technologies, separately from the repository origin. Another example: If a repository of professionally managed software contains components that are categorized as exploratory (a mix), the entire repository is still categorized as professional, and decisions must be filtered at a different level.

As this categorization may be use case specific, this should be considered as a limitation of the work for a general statement.

## 9 CONCLUSION

In a novel approach to categorize software repositories as exploratory or professional based on their

README.md texts, we have established a detailed definition and annotated a new dataset of over 200 repositories from GitHub. Our evaluation of various approaches reveals that LLMs generally outperform classical NLP approaches in classifying software repositories based on README files. OpenAI’s GPT4o model achieved the best zero-shot classification results without multi-step reasoning, while Google’s Gemini 1.5 Flash showed strong performance among the smaller, cost-efficient models. The few-shot approach improved the accuracy of some LLMs, notably Llama 3 (70b) and Claude 3.5 Sonnet, achieving the highest accuracy of 89.5% with multi-step reasoning using Llama 3. Filtering based on word probability thresholds showed mixed results, with GPT4o-Mini benefiting from this method, equal to Gemini 1.5 Flash. However, the tested approaches, such as few-shot and multi-step reasoning, did not consistently lead to better results, which is why the methodology must be evaluated according to the use case.

Time and cost considerations also highlighted that smaller models and direct prompt-based queries led to faster, more cost-effective classification, especially when reasoning steps were omitted. Overall, the study demonstrates that classification of README-based repositories is feasible on around 70% of the data, which was assignable.

## 10 FUTURE WORK

We showed that it is possible to classify a major part of software repositories based on the README files, but there are still challenges to overcome. One of the main challenges is the lack of information in many README files. To overcome this challenge, we plan to explore other sources of information, such as the number of stars, forks, watches, issues, commits, and other things, like code comments, commit messages, licensing and issue discussions. We also plan to investigate other approaches, such as fine-tuning the Llama 3 model to optimize the classification results even further. Another challenge is the lack of annotated data. We therefore plan to extend the dataset by adding more data and including more annotators to reduce bias.

This study was conducted with the aim of automatically extracting design and architecture decisions from existing software repositories and making them available for a recommendation system or as a database for an LLM. Further studies on the research project are planned.

## REFERENCES

Capiluppi, A., Di Ruscio, D., Di Rocco, J., Nguyen, P. T., and Ajienska, N. (2020). Detecting java software similarities by using different clustering techniques. *Information and Software Technology*, 122:106279.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. (2024). Retrieval-augmented generation for large language models: A survey.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pre-training approach. *arXiv preprint arXiv:1907.11692*.

Petrovic, G., Dimitrieski, V., and Fujita, H. (2016). A deep learning approach for searching cloud-hosted software projects. In *SoMeT*, pages 358–368.

Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., and Lo, D. (2019). Categorizing the content of github readme files. *Empirical Software Engineering*, 24:1296–1327.

Rao, A. E. and Chimalakonda, S. (2022). Apples, oranges & fruits – understanding similarity of software repositories through the lens of dissimilar artifacts. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 384–388.

Selva Birunda, S. and Kanniga Devi, R. (2021). A review on word embedding techniques for text classification. In Raj, J. S., Iliyasu, A. M., Bestak, R., and Baig, Z. A., editors, *Innovative Data Communication Technologies and Application*, pages 267–281, Singapore. Springer Singapore.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Xu, W., Sun, X., Hu, J., and Li, B. (2017). Repersp: Recommending personalized software projects on github. pages 648–652.

Zhang, Y., Lo, D., Kochhar, P. S., Xia, X., Li, Q., and Sun, J. (2017). Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23.

## APPENDIX

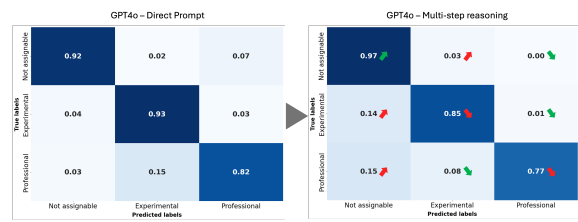


Figure 6: Changes in classification by OpenAI GPT4o with direct and multi-reasoning step by comparison of normalized CMs.

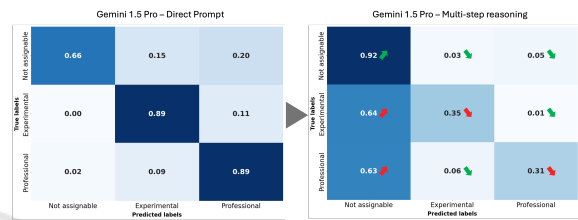


Figure 7: Changes in classification by Google Gemini 1.5 with direct and multi-reasoning step by comparison of normalized CMs.

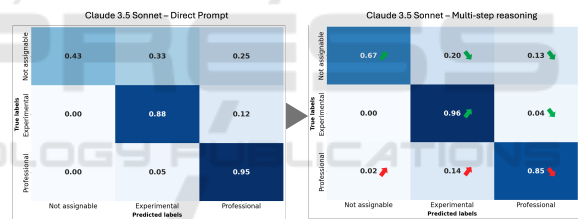


Figure 8: Changes in classification by Anthropic Claude 3.5 Sonnet with direct and multi-reasoning step by comparison of normalized CMs.