CENG 518 Introduction to Research Methodology And Ethics Project Proposal LLM-Based Design Pattern Detection

Gökay Gülsoy[1] 323078045

İzmir Institute of Technology, Computer Engineering Department

1 Introduction

Design patterns (DPs) are standardized, reusable approaches to common software design problems. The use of design patterns is considered a best practice for improving the quality of software products by enhancing their maintainability, elegance, flexibility, and understandability [1]. Because of the benefits they offer and the error-prone nature of manual inspection, automatic design pattern recognition has emerged as a key area of reverse engineering research, helping to automatically identify the use of design patterns. It improves the understanding of design decisions and assists in the process of software re-documentation, re-implementation, and reuse.

2 Objectives of the Project

Design Patterns (DPs) are essential in software development; however, manually identifying and categorizing patterns in large, complex code bases can be time-consuming. This project aims to develop a natural language processing system utilizing publicly available large language models that can provide automatic detection of DPs implemented in Java programming language to aid developers in quickly and accurately understanding and maintaining unfamiliar source code, ultimately enhancing productivity.

3 Related Works

Various approaches have been proposed to address this challenge. One of the primary studies, such as Code2Vec [2], demonstrates remarkable success in various programming tasks and outperforms traditional word-based embedding models by representing code snippets as continuous distributed vectors rather than raw words, and therefore enhancing the distinctiveness of the code snippets in the vector space. Xiao et al. Matching-based approaches are widely used, such as those that rely on similarity score [3] or graph structures [4]. Moreover, multi-stage approaches have been explored. These usually involve a learning phase [5,6]

or pattern definition phase [7,8] prior to the actual detection of design patterns. Learning-based approaches in the first phase are limited because they require a significant amount of annotated training data. This data has to encompass the necessary diversity of design pattern instances for the patterns of interest. Formal definitions of design pattern structures also pose limitations. These include the expressiveness of the choice of language, the mapping of abstract concepts to language-specific constructs, and the inherent difficulty of defining patterns at this level of abstraction. Several approaches in the literature focus on classifying source code as an instance of a design pattern using fixed-length inputs, such as a single class [9,10]. These methods typically work well for small or well-defined code segments where the scope is narrowly confined. However, they might fail to capture the broader context of how multiple classes or modules interact, leading to incomplete representations of the software.

4 Methodology

The proposed methodology provides a novel approach leveraging large language models to automatically identify creational design pattern instances across diverse codebases. The dataset named P-mart that contains the design pattern instances to be used is published by [11], which is a well-known dataset for the design pattern classification problem in the software engineering community, and in addition to annotating the design pattern instances, the authors also provided specific versions of the related source code. Methodology followed in this project consists of 4 main steps, which are parallel to those followed by [12].

First, during the data preparation phase, available annotated design pattern instances are thoroughly examined to ensure their suitability for analysis. As a part of this process, instances for which all the associated source code files are not accessible are filtered out, as incomplete datasets could compromise the reliability of subsequent steps. Then, data cleaning is done to ensure that formatting inconsistencies do not introduce errors or ambiguities in the later stages. In the second phase, prompts are prepared by providing the source code of one example from the necessary sources using PDF extraction and web scraping techniques, along with the corresponding annotation of the pattern. In a second prompt message, the source code snippet of the second sample is provided.

In order to determine which classes to include in the provided example snippet and in the snippet for which the design pattern needs to be identified, the root package shared by all classes participating in the ground truth annotations is identified. In the third phase prediction for the given example is performed using selected publicly available large language models (LLMs) by providing each prompt to a large language model and generating output in a structured format, then storing the response alongside the respective example and design pattern expected to be found. In the final fourth phase, responses provided by the large language model and structured outputs are analyzed. In cases where the large language model does not provide any annotations, it responds with

a clear answer, which is treated as no design pattern instance identified within the analyzed context. Finally, Classification results are evaluated with accuracy, precision, recall, and F1-score metrics to interpret and assess the performance of the overall system.

5 Requirements

As mentioned in the methodology section, the P-mart dataset, which contains design pattern instances, is required for annotated source code examples and the classification task. Besides, the Python programming language needs to be used as it provides the necessary libraries for natural language processing tasks and to utilize LLMs with the langchain framework.

References

1. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
2. U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," 2018.
3. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *IEEE transactions on software engineering*, vol. 32, no. 11, pp. 896–909, 2006.
4. B. B. Mayvan and A. Rasoolzadegan, "Design pattern detection based on the graph theory," *Knowledge-Based Systems*, vol. 120, pp. 211–225, 2017.
5. N. Bozorgvar, A. Rasoolzadegan, and A. Harati, "Probabilistic detection of gof design patterns," *The Journal of Supercomputing*, vol. 79, no. 2, pp. 1654–1682, 2023.
6. R. Barbudo, A. Ramírez, F. Servant, and J. R. Romero, "Geml: A grammar-based evolutionary machine learning approach for design-pattern detection," *Journal of Systems and Software*, vol. 175, p. 110919, 2021.
7. G. Rasool and P. Mäder, "Flexible design pattern detection based on feature types," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 243–252, IEEE, 2011.
8. H. Thaller, L. Linsbauer, and A. Egyed, "Feature maps: A comprehensible software representation for design pattern detection," in *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*, pp. 207–217, IEEE, 2019.
9. N. Nazar, A. Aleti, and Y. Zheng, "Feature-based software design pattern detection," *Journal of Systems and Software*, vol. 185, p. 111179, 2022.
10. R. Mzid, I. Rezgui, and T. Ziadi, "Attention-based method for design pattern detection," in *European Conference on Software Architecture*, pp. 86–101, Springer, 2024.
11. Y.-G. Guéhéneuc, "P-mart: Pattern-like micro architecture repository," *Proceedings of the 1st EuroPLoP Focus Group on pattern repositories*, pp. 1–3, 2007.
12. C. Schindler and A. Rausch, "Llm-based design pattern detection," *arXiv preprint arXiv:2502.18458*, 2025.