Large Language Model Based Design Pattern Detection

Gökay Gülsoy 323078045

CENG 518 Introduction to Research Methodology and Ethics
İzmir Institute of Technology

December 25, 2025

Table of Contents

What is a Design Pattern ?

- Design patterns (DP) 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, flexibility, and understandability [1].
- The Well-known book Design Patterns: Elements of Reusable Object-Oriented Software [1] lists most widely used 23 design patterns.

Categories of Design Patterns

- Behavioral
  1. Interpreter
  2. Template Method
  3. Chain of Responsibility
  4. Command
  5. Iterator
  6. Mediator
  7. Memento
  8. Observer
  9. State
  10. Strategy
  11. Visitor
- Creational Patterns
  1. Factory Method
  2. Abstract Factory
  3. Builder
  4. Prototype
  5. Singleton
- Structural Patterns
  1. Adapter
  2. Bridge
  3. Composite
  4. Decorator
  5. Facade
  6. Flyweight
  7. Proxy

Table of Contents

Inspecting and Detecting Design Patterns

- Because of the benefits they offer and the error-prone nature of manual inspection, automatic design pattern recognition emerged as a research area.
- It enhances the understanding of design decisions and facilitates the process of software re-documentation, re-implementation, and reuse.

Table of Contents

Objectives of Study

- Design patterns are essential in software development; however, manually identifying patterns in large, complex code bases can be time-consuming.

Objectives of Study

- Design patterns are essential in software development; however, manually identifying patterns in large, complex code bases can be time-consuming.
- This study aims to develop a natural language processing approach utilizing publicly available large language models, which can be utilized in the automatic detection of DPs implemented in Java programming language.

Objectives of Study

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

Table of Contents

What are the studies that have already been done ?

Various approaches have been proposed to address this challenge:

- Code2Vec [2] demonstrates remarkable success in programming tasks and outperforms traditional word-based embedding models by representing code snippets as continuous distributed vectors rather than raw words.

What are the studies that have already been done ?

Various approaches have been proposed to address this challenge:

- Code2Vec [2] demonstrates remarkable success in programming tasks and outperforms traditional word-based embedding models by representing code snippets as continuous distributed vectors rather than raw words.
- Matching-based approaches are widely used, such as those based on a similarity score [3] or graph structures [4].

What are the studies that have already been done ?

Various approaches have been proposed to address this challenge:

- Code2Vec [2] demonstrates remarkable success in programming tasks and outperforms traditional word-based embedding models by representing code snippets as continuous distributed vectors rather than raw words.
- Matching-based approaches are widely used, such as those based on a similarity score [3] or graph structures [4].
- Multi-stage approaches have also been explored. These usually involve a learning phase [5] or a pattern definition phase [6] prior to actual detection of design patterns.

Limitations of Existing Approaches

- Learning-based approaches are limited because they require a significant amount of annotated training data, and these data have to encompass the necessary diversity of design pattern instances for the patterns of interest.

Limitations of Existing Approaches

- Learning-based approaches are limited because they require a significant amount of annotated training data, and these data have to encompass the necessary diversity of design pattern instances for the patterns of interest.
- 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 [7].

Table of Contents

Proposed Methodology

- The proposed methodology provides a novel approach using large language models to automatically identify creational design pattern instances across diverse codebases.

Proposed Methodology

- The proposed methodology provides a novel approach using large language models to automatically identify creational design pattern instances across diverse codebases.
- Dataset to be used is P-mart [8] that contains the design pattern instances for the design pattern classification problem in the software engineering community.

Proposed Methodology

- The proposed methodology provides a novel approach using large language models to automatically identify creational design pattern instances across diverse codebases.
- Dataset to be used is P-mart [8] that contains the design pattern instances for the design pattern classification problem in the software engineering community.
- Methodology followed in this project consists of 4 main steps.

Phases of Followed Methodology

First Phase:

- The data preparation phase, in which annotated design patterns are thoroughly examined to ensure their suitability for analysis.

Phases of Followed Methodology

First Phase:

- The data preparation phase, in which annotated design patterns are thoroughly examined to ensure their suitability for analysis.
- As a part of this process, instances for which all the associated source code files that are not accessible are filtered out, as incomplete datasets could compromise the reliability of subsequent steps.

Phases of Followed Methodology

First Phase:

- The data preparation phase, in which annotated design patterns are thoroughly examined to ensure their suitability for analysis.
- As a part of this process, instances for which all the associated source code files that are not accessible are filtered out, as incomplete datasets could compromise the reliability of subsequent steps.
- Then data cleaning is performed to ensure that formatting inconsistencies do not introduce errors or ambiguities in the later stages.

Phases of Followed Methodology Cont.

Second Phase:

- Prompts are prepared by providing the source code of one example from the P-mart dataset, along with the corresponding annotation of the pattern.

Phases of Followed Methodology Cont.

Second Phase:

- Prompts are prepared by providing the source code of one example from the P-mart dataset, along with the corresponding annotation of the pattern.
- In a second prompt message, the source code snippet to be classified is provided from a different project included in the P-mart dataset.

Phases of Followed Methodology Cont.

Second Phase:

- Prompts are prepared by providing the source code of one example from the P-mart dataset, along with the corresponding annotation of the pattern.
- In a second prompt message, the source code snippet to be classified is provided from a different project included in the P-mart dataset.
- In order to determine which classes to include in the provided example snippet and for which the design pattern needs to be identified, the root package shared by all classes participating in the ground truth annotation is identified.

Phases of Followed Methodology Cont.

Third Phase:

- Prediction for the given example is performed using selected publicly available large language models (LLMs) by providing each prompt to the large language model and generating output in a structured format.

Phases of Followed Methodology Cont.

Third Phase:

- Prediction for the given example is performed using selected publicly available large language models (LLMs) by providing each prompt to the large language model and generating output in a structured format.
- The response is stored along with the respective example and the design pattern that is expected to be found.

Phases of Followed Methodology Cont.

Fourth Phase:

- Response provided by the large language model and structured outputs are analyzed.

Phases of Followed Methodology Cont.

Fourth Phase:

- Response 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.

Phases of Followed Methodology Cont.

Fourth Phase:

- Response 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.

LLM-based Design Pattern Detection Work Flow

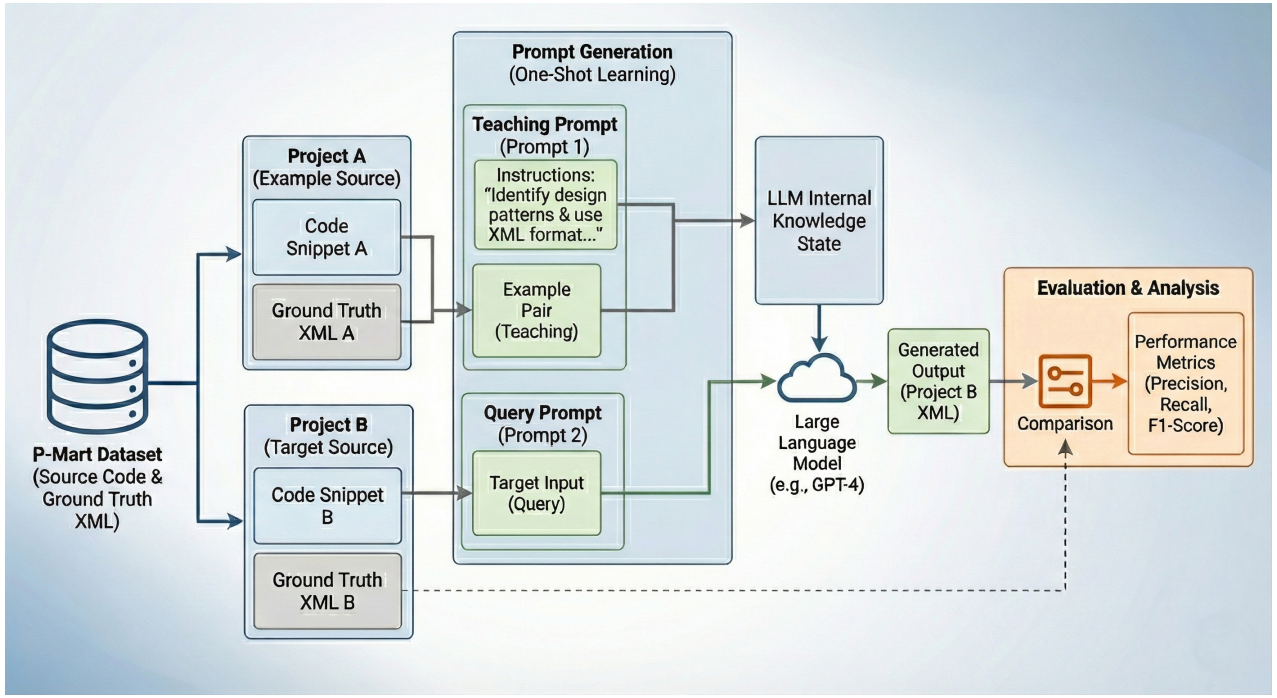

Figure: LLM-based Design Pattern Detection

[1] Figure created using Gemini Nano Banana Pro

Table of Contents

Requirements

- P-mart dataset, which contains design pattern instances, is required for annotated source code examples and the classification task.
- 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.

Table of Contents

Questions ?

[1] Image taken from: https://business.tutsplus.com/tutorials/thank-you-slide-powerpoint-presentation--cms-34231

Table of Contents

References I


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.


U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," 2018.


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.

References II

- B. Bafandeh Mayvan and A. Rasoolzadegan, "Design pattern detection based on the graph theory," *Know.-Based Syst.*, vol. 120, p. 211–225, Mar. 2017.
- 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.
- 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, 2011.

References III


N. Nazar, A. Aleti, and Y. Zheng, "Feature-based software design pattern detection," *J. Syst. Softw.*, vol. 185, Mar. 2022.


Y.-G. Guéhéneuc, "P-mart: Pattern-like micro architecture repository," *Proceedings of the 1st EuroPLoP Focus Group on pattern repositories*, pp. 1–3, 2007.