PATTERNS 2026

The Eighteenth International Conferences on Pervasive Patterns and Applications

ISBN: 978-1-68558-378-1

April 19 - 23, 2026

Lisbon, Portugal

PATTERNS 2026 Editors

Steve Chan, Decision Engineering Analysis Laboratory, USA

PATTERNS 2026

Forward

The Eighteenth International Conferences on Pervasive Patterns and Applications (PATTERNS 2026), held on April 19 – 23, 2026, continued a series of events targeting the application of advanced patterns, at-large. In addition to support for patterns and pattern processing, special categories of patterns covering ubiquity, software, security, communications, discovery and decision were considered. It is believed that patterns play an important role on cognition, automation, and service computation and orchestration areas. Antipatterns come as a normal output as needed lessons learned.

The conference had the following tracks:

- Patterns basics
- Patterns at work
- Discovery and decision patterns
- Medical and facial image patterns
- Tracking human patterns

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the PATTERNS 2026 technical program committee, as well as the numerous reviewers. The creation of a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to PATTERNS 2026. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the PATTERNS 2026 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope PATTERNS 2026 was a successful international forum for the exchange of ideas and results between academia and industry that will promote further progress in the area of pervasive patterns and applications. We also hope that Lisbon provided a pleasant environment during the conference and everyone saved some time to enjoy this beautiful city.

PATTERNS 2026 Steering Committee

Herwig Manaert, University of Antwerp, Belgium
Wladyslaw Homenda, Warsaw University of Technology, Poland

PATTERNS 2026 Publicity Chair

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain
Ali Ahmad, Universitat Politècnica de València, Spain
José Miguel Jiménez, Universitat Politècnica de València, Spain
Sandra Viciano Tudela, Universitat Politècnica de València, Spain

PATTERNS 2026

Committee

PATTERNS 2026 Steering Committee

Herwig Manaert, University of Antwerp, Belgium
Wladyslaw Homenda, Warsaw University of Technology, Poland

PATTERNS 2026 Publicity Chair

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain
Ali Ahmad, Universitat Politècnica de València, Spain
José Miguel Jiménez, Universitat Politècnica de València, Spain
Sandra Viciano Tudela, Universitat Politècnica de València, Spain

PATTERNS 2026 Technical Program Committee

Andrea F. Abate, University of Salerno, Italy
Akshay Agarwal, IIIT Delhi, India
Carlos Alexandre Ferreira, INESC TEC, Portugal
Vijayan K. Asari, University of Dayton, USA
Danilo Avola, Sapienza University of Rome, Italy
Johanna Barzen, University of Stuttgart, Germany
Frederik Simon Bäumer, Bielefeld University of Applied Sciences, Germany
Martin Beisel, University of Stuttgart, Germany
Nadjia Benblidia, Saad Dahlab University - Blida1, Algeria
Anna Berlino, Consultant in Tourism Sciences and Valorization of Cultural and Tourism Systems, Italy
Fatma Bouhlel, University of Sfax, Tunisia
Uwe Breitenbücher, Reutlingen University, Germany
Alceu S. Britto, Pontifical Catholic University of Paranā (PUCPR), Brazil
Eliot Bytyçi, University of Prishtina "Hasan Prishtina", Kosovo
Isaac Caicedo-Castro, University of Córdoba, Colombia
Simone Cammarasana, CNR-IMATI, Genova, Italy
David Cárdenas-Peña, Universidad Tecnológica de Pereira, Colombia
Bidyut B. Chaudhuri, Indian Statistical Institute, India
Sneha Chaudhari, AI Organization | LinkedIn, USA
Diego Collazos, Universidad Nacional de Colombia sede Manizales, Colombia
Sergio Cruces, University of Seville, Spain
Mohamed Daoudi, Institut Mines-Telecom / Telecom Lille, France
Jacqueline Daykin, King's College London, UK / Aberystwyth University, Wales & Mauritius
Moussa Diaf, Mouloud Mammeri University, Algeria
Chawki Djeddi, Université de Tébessa, Algeria
Ole Kristian Ekseth, NTNU & Eltorque, Norway

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

On Evolvability in Tools for Data Analytics and Intelligence: the Power BI case 1
Geert Haerens and Herwig Mannaert

A Methodology for Investigating AI Patterns Prevalence in Software Repositories 7
Srinath Perera, Hasinthaka Piyumal, Frank Leymann, and Rania Khalaf

On Evolvability in Tools for Data Analytics and Intelligence: the Power BI case

Geert Haerens
Antwerp Management School, Belgium
Engie nv, Belgium
Email: geert.haerens@engie.com

Herwig Mannaert
University of Antwerp, Belgium
Email: herwig.mannaert@uantwerpen.be

***Abstract*—Data analytics and business intelligence are essential to contemporary organizations. Tools, such as Microsoft Power BI are widely used and appreciated by businesses because they offer low-code/no-code features, putting data analytics capabilities in the hands of those who best know the data. A closer analysis of analytics products built with Power BI's low-code/no-code features reveals significant evolvability issues that can only be resolved by leaving the low-code/no-code path and adopting a coding approach. In this paper, we will analyze the evolvability issues using Normalized Systems theory and argue that the low-code/no-code features produce non-evolvable solutions.**

Keywords*—*Normalized Systems Theory; Evolvability; Low-Code/No-Code; Power BI.

I. Introduction

Data analytics, intelligence, and visualization used to be in the hands of IT, where businesses depended upon IT to provide them with views on their data, as businesses lacked the skills to exploit and use such solutions. Today, data analytics is made available to businesses via easy-to-use low-code/no-code solutions. You no longer need to be a technical expert to process, analyze and visualize your data. Click/configure gestures have replaced Structured Query Language (SQL) statements and other coding techniques, thereby democratizing Business Intelligence (BI) capabilities.

We note, however, that the "easy-to-learn Graphical User Interface (GUI)" (i.e., low-code/no-code) is not yielding evolvable solutions. We use the concept of evolvability as defined by Normalized Systems theory (NS) (see Section III) and observe that simple changes, such as adding or removing data from the source, can break the solution when the "easy-to-learn GUI" is used. Those issues can be resolved, but then we leave the low-code/no-code path and begin developing code.

Based on that observation, the main argument of this paper is that the low-code functionalities deliver non-evolvable solutions, unless you start to code. But what is the point of low-code/no-code in that case? As such, the paper highlights an anti-pattern.

In this paper, we use Microsoft Power BI as a case study to further explore our observations. This paper is structured as follows. We start by introducing the data analytics and intelligence tool, Power BI, in Section II that we will use to build our case, followed by an introduction to NS theory in Section III. Section IV presents an example that demonstrates evolvability issues in Power BI, and Section V reflects on their impact. The paper is wrapped up in Section VI.

To the best of our abilities, we have not found similar case studies, and we decided not to include a related work section. Instead, we have incorporated relevant literature where applicable across the sections.

II. Data Analytics and Intelligence: the Power BI Tool

Data analytics, intelligence, and visualisation are a billion-dollar market. An objective source stating the exact market value is difficult to find. The available data mostly comes from market research firms, published in pay-for-play analysis reports. A quick Google search, well aware of its low academic value, indicates a market value of approximately $10 billion and a projected increase of 50-100% by 2030 [1]- [3].

Power BI is a leader in Gartner's Magic Quadrant for Analytics and Business Intelligence Platforms [4]. In the same report, the data preparation capabilities of Power BI via Power Query are cited as its strongest feature. In [5], Gartner describes this capability as:

"Microsoft Power BI's strongest capability is data preparation. Its Power Query feature enables data analysts to blend disparate datasets via an easy-to-learn GUI and automatically detects data types and relationships. Columns are assigned a default data type during load. Business users can override the automatic relationship in the visual diagram model or relationships menu."

Other lauded capabilities are the creation of metics, the integration with Microsoft's data ecosystem, the inclusion of GenAI via Copilot, and its market presence and talent availability.

According to Microsoft [6]: "Power BI is Microsoft's business analytics platform that helps you turn data into actionable insights. Whether you're a business user, report creator, or developer, Power BI offers integrated tools and services to connect, visualize, and share data across your organization."

Power BI is part of the Microsoft Fabric data platform. The version of Power BI used by desktop users is called Power BI Desktop. Power BI Desktop allows you to connect to different data sources, transform the data as required, and visualize it on a dashboard using various visualization widgets (e.g., tables, bar charts, pie charts). A dashboard created in Power BI Desktop can be published and shared in Power BI Services. Power BI Desktop is used to create personal dashboards and publish them to Power BI Services. Power BI Desktop contains two subtools: Power Query and Power BI Desktop (confusingly having the same name as the main product).

 ISBN: 978-1-68558-378-1

The function of Power Query is to extract, transform and load (aka ETL [7]) data coming from different sources into a data format/store that can be used in Power BI Desktop to visualize and analyze the data. Data is read from a source and loaded into a tabular representation called a query. For that query, various transformations and filters can be applied via low-code functionality. Examples of transformations on the columns of a query include renaming, deleting, reordering, adding columns, and changing data types. Examples of filters include sorting and filtering on text, numerical patterns, or expressions. Queries can also be merged, pivoted and anti-pivoted. We refer to [8] for more elaborate analysis on how low-code can be used to perform ETL.

The function of Power BI Desktop is to visualize the data that is exposed by Power Query. The data is provided in tabular form and can be linked using the standard Entity-Relationship Model features, thereby creating a data model. The data can be sliced and diced via filters and filtering functionalities of the different visualization widgets. The dashboard can then be published in the Power BI Services environment and shared with other users.

Interaction with both tools is mediated by low-code capabilities. This is a key feature of Power BI and an important value proposition of the tool: anybody can do Business Intelligence - you don't need to be an IT expert. The user does not need to write code; they only need to configure transformation steps in Power Query and filters and widgets in Power BI Desktop. Both components have a programming language under the hood. In Power Query, this is M, a language for working with queries and lists; in Power BI Desktop, this is DAX, a language similar to the functions available in Excel.

III. Fundamentals of NS theory

Software should be able to evolve as business requirements change over time. In NS theory [9] [10], the lack of Combinatorial Effects (CE) measures evolvability. When the impact of a change depends not only on the type of change but also on the size of the system it affects, we refer to a CE. The NS theory assumes that software grows indefinitely and undergoes unlimited changes over time. Under those conditions, CEs harm software evolvability.

NS theory is built on the principles of classical system engineering and statistical entropy. In classical system engineering, a system is stable if it has bounded-input, bounded-output (BIBO) stability. NS theory applies this idea to software design: a limited change in functionality should produce a limited change in the software. In classic system engineering, stability is measured at infinity. NS theory considers infinitely large systems that undergo an infinite number of changes. A system is stable for NS if it does not have CE, meaning that the effect of change only depends on the kind of change and not on the system size.

NS theory posits four theorems and a method for enforcing them through software extension. The theorems are proven formally, yielding a set of required conditions that must be strictly followed to avoid CE.

A. NS Theorems

NS theory [9] is based on four theorems that dictate the necessary conditions for software to be free of CE.

- Separation of Concerns (SoC)
- Data Version Transparency (DvT)
- Action Version Transparency (AvT)
- Separation of States (SoS)

Violating any of these four theorems will lead to CE and, thus, non-evolvable software under change.

Consistently adhering to the four NS theorems is challenging for developers. NS theory proposes the use of software expansion mechanisms, where software templates, called elements, are manually crafted with close attention to NS theorems, allowing reusage by combining a generalized pattern and parameter-based input. The expanded elements can then be customized to meet the required specifications. All customizations can be collected via a process called harvesting, allowing the independent evolution of patterns and customizations via the rejuvenation process. More information on this harvesting and rejuvenation can be found in [11].

B. NS and Power BI

In the context of this paper's topic, we want to investigate the impact of changes to the source data that is consumed by Power BI, and see the effect this has on the configured ETL steps in Power Query, and the impact on data visualizations in Power BI Desktop. We also want to investigate the effect of making changes to the visualizations in Power BI Desktop.

More concretely, for the ETL part, we are interested in Power BI's compliance with DvT and AvT. Dvt and AvT are defined as follows:

- DvT: A data structure that is passed through the interface of a processing function needs to exhibit version transparency in order to achieve stability.
- AvT: A processing function that is called by another processing function needs to exhibit version transparency in order to achieve stability.

The processing function will be an ETL step, and the version transparency indicates that execution should continue to work and produce the expected result when the input data changes.

The simplest change is adding new data attributes to a data item. A processing step configured to operate on all data attributes of a data item should also process the new data attributes. Likewise, a processing function that should process only selected data attributes of a data item should still be able to process those attributes even if additional attributes are added to the data item.

Compliance with the NS theorems assures we don't introduce CE for changes that add something. It does not, directly, protect against the removal of something. If the source data contains less data than is functionally expected, this processing function will indeed break. Note the importance of "functionally expected". If the functionality is "I want to perform operation X on data attribute Y of data item Z", and data attribute Y is not in the source, it is normal for the task

to fail. If the functionality is ”I want to perform operation X on all data attributes of data item Z”, then the task should continue to work independently of the number of attributes of data item Z. Thus, the impact of a removal depends on the functionality it entails. Compliance with the NS theorems ensures that no CE are introduced when processing functions no longer require a certain data attribute. They should be able to ignore data attributes they do not perform actions on.

For the data visualization part of Power BI, we want to investigate the impact of adding data widgets to the dashboard canvas. As it is known that visualization widgets can interact with each other on the dashboard canvas, we will be looking more carefully at SoC.

- SoC: A processing function can only contain a single task in order to achieve stability.

The processing function will be a vizualization function in this case, and the tasks can be the filtering, configuration and interactions. As multiple tasks are linked to a visualization function, there is a high probability that CE will be introduced. In Section IV, we will test these scenarios to evaluate NS compliance of Power BI.

IV. Demonstrating Evolvability Issues in Power BI

As outlined in Section II, Power BI has two main components: Power Query (used for data transformations) and Power BI Desktop (used for data visualisation). In this section, we demonstrate that both components exhibit evolvability issues and argue that these issues could have been avoided. In Section IV-A, we will discuss a well-known evolvability issue in Power Query, and in Section IV-B, we will discuss a less recognized evolvability issue in Power BI Desktop.

A. Evolvability Issues in Power Query

In Power Query, a source $\mathbf{S}$ is transformed into $\mathbf{S_T}$ by means of a transformation function $\mathbf{T}$. $\mathbf{T}$ applies consecutive transformations $\mathbf{T_i}$ on $\mathbf{S}$, where the output of $\mathbf{T_i}$ becomes the input for $\mathbf{T_{i+1}}$.

$$\mathbf{S_T} = \mathbf{T}(\mathbf{S}), \qquad \mathbf{T} = \mathbf{T_n} \circ \mathbf{T_{n-1}} \circ \cdots \circ \mathbf{T_1}$$

The different transformations $\mathbf{T_i}$ are configured via Power Query’s low-code functionality: select a query/column/row, choose a transformation method from the menu, and provide additional transformation parameters. Each transformation automatically gets a name assigned and is visualized in the query properties plane. Power Query translates each configured transformation step into an M statement. The consecutive execution of the different M-statements constitutes the total query transformation $\mathbf{T}$. M is a proprietary Microsoft ETL programming language. Its features are rich and powerful, but limited to the Microsoft ecosystem.

Let us consider the following example: We want to create a basic ETL cleanup function $\mathbf{T}$ that takes all data from a source, puts it in a tableau representation, assigns column names based on the source data’s first row, treats all incoming data as text, and replaces empty and null values with a default value of **None**. We need our transformation function to anticipate changes to the source data, such as the addition or removal of columns, or the renaming of columns.

We use an Excel file as a source (see Figure 1). The file contains six columns, the first three are text, the fourth is a decimal number, the fifth is a date, and the last is a percentage. We use only Power Query’s low-code functionality to perform the necessary transformations. The amount of data is limited, as our objective is not to evaluate performance but evolvability.

The example is a relevant use case: a non-IT person who lacks coding skills wants all source files uniformly transformed, and they would use Power BI’s low-code functionality.

- Step 1: Select sheet 1 of the file = ”source.xlsx” as source data
- Step 2: Promote the first row of the source to column headers
- Step 3: Select all columns and transform the data type to ”Text”
- Step 4: Select all columns and replace the empty string with ”**None**”
- Step 5: Select all columns and replace the null character with ”**None**”

Name	Description	Business Criticality	CAPEX (2023)	Lifecycle: Active	Completion
ABC123	some random text 1	businessOperational		2020-01-01	87%
ABC124	some random text 2	businessCritical		2020-01-01	100%
ABC125	some random text 3	businessOperational			68%
ABC126	some random text 4	businessOperational	100	2020-01-01	62%
ABC127	some random text 5	missionCritical		2020-01-01	62%
ABC128	some random text 6	businessOperational	50	2022-01-01	68%
ABC129	some random text 7			2025-03-13	100%
ABC130	some random text 8	businessOperational	0	2020-01-01	62%
ABC131	some random text 9	businessOperational		2022-01-01	68%
ABC132	some random text 10	businessOperational		2020-01-01	62%
ABC133	some random text 11	administrativeService	3,3	2022-01-01	68%
ABC134	some random text 12	businessOperational	20	2022-01-01	68%

Fig. 1. Source file.

The applied steps and the result can be seen in the Query Properties plane (see Figure 2).

We now test if our transformation respects DvT and AvT. We add a new column to our source.xlsx file (a copy of the ”CAPEX (2023)” - see Figure 3) column, and we apply the same transformations. The result of the transformation is shown in Figure 4. We observe that, after the transformation, the new column is not the same data type as the others and contains empty strings.

We now rename a column in our source.xlsx file (rename ”CAPEX (2023)” to ”CAPEX (2025) - see Figure 5) and apply the same transformations. The result of the transformation is shown in Figure 6. We notice that the transformation produces an error.

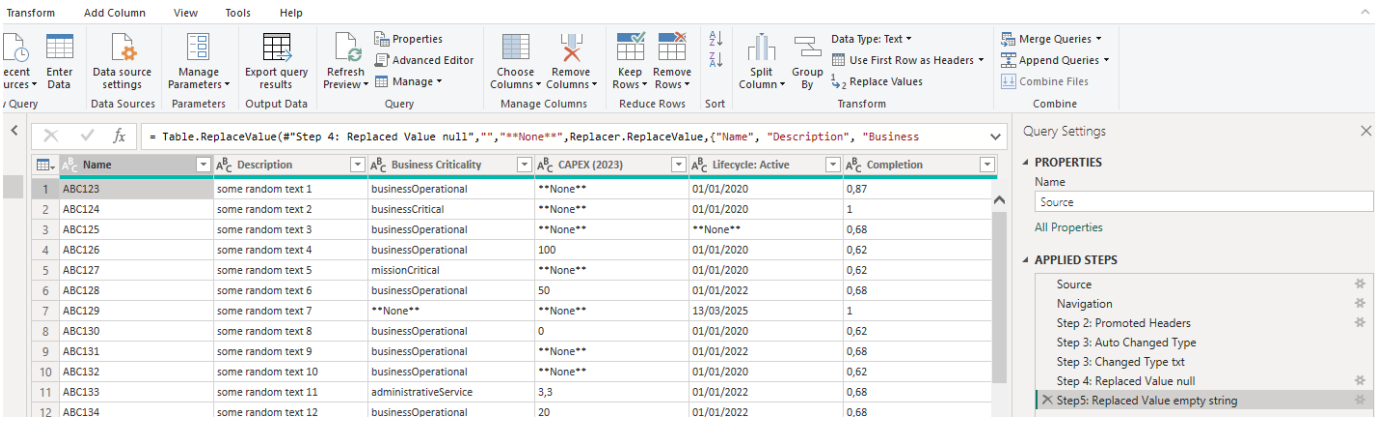

Fig. 2. Transformation Steps 1 to 5..

We conclude that the transformation function created in Power Query does not adhere to DvT and AvT, as changes to the data structure yield unexpected output.

Let us examine the M code generated by Power Query from our low-code configurations (see Figure 7). The M statement responsible for converting to the data type ”text” (line 6 in Figure 7) explicitly lists column names that match those in the initial data source. Logically, new columns are not accounted for, and if columns no longer exist or have been renamed, the transformation fails. The M statements responsible for the replacements of the empty string and null character (lines 7 and 8 in Figure 7) also contain an explicit list of column names that match the initial data source. Again, new columns are not accounted for, and if columns no longer exist or are renamed, the transformation fails.

Name	Description	Business Criticality	CAPEX (2023)	Lifecycle: Active	Completion	CAPEX (2023) 2
ABC123	some random text 1	businessOperational		2020-01-01	87%	
ABC124	some random text 2	businessCritical		2020-01-01	100%	
ABC125	some random text 3	businessOperational			68%	
ABC126	some random text 4	businessOperational	100	2020-01-01	62%	100
ABC127	some random text 5	missionCritical		2020-01-01	62%	
ABC128	some random text 6	businessOperational	50	2022-01-01	68%	50
ABC129	some random text 7			2025-03-13	100%	
ABC130	some random text 8	businessOperational	0	2020-01-01	62%	0
ABC131	some random text 9	businessOperational		2022-01-01	68%	
ABC132	some random text 10	businessOperational		2020-01-01	62%	

Fig. 3. Adding extra column to source.

This issue is well-documented and recognized, even by Microsoft [12].

The M language provides a statement that returns the list of columns for a query. If the transformation function had first included this statement and used the result for Steps 3 to 5, we would have created a transformation function compliant with our requirements and thus data-version transparent. We will discuss this in more detail in Section V. The problem is, we can only do this by editing the M code directly. There is no low-code button/click/drag-drop or parameter available to configure the use of this statement.

Using Power Query’s low-code column transformation functionality, the initial source column names are preserved,

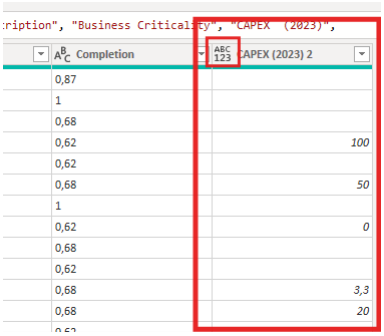

Fig. 4. Result of the transformation after adding a column.

Name	Description	Business Criticality	CAPEX (2025)	Lifecycle: Active	Completion	CAPEX (2023) 2
ABC123	some random text 1	businessOperational		2020-01-01	87%	
ABC124	some random text 2	businessCritical		2020-01-01	100%	
ABC125	some random text 3	businessOperational			68%	
ABC126	some random text 4	businessOperational	100	2020-01-01	62%	100
ABC127	some random text 5	missionCritical		2020-01-01	62%	
ABC128	some random text 6	businessOperational	50	2022-01-01	68%	50
ABC129	some random text 7			2025-03-13	100%	
ABC130	some random text 8	businessOperational	0	2020-01-01	62%	0
ABC131	some random text 9	businessOperational		2022-01-01	68%	
ABC132	some random text 10	businessOperational		2020-01-01	62%	
ABC133	some random text 11	administrativeService	3,3	2022-01-01	68%	3,3

Fig. 5. Renaming a column in the source.

violating DvT and making the column transformation non-evolvable. However, the M language does include the required statements and functions to perform column transformations using DvT, but you need to code them explicitly. This leads us to conclude that Power Query’s low-code functionality, by default, produces non-evolvable transformation functions. To get evolvable transformation functions, you need to code,

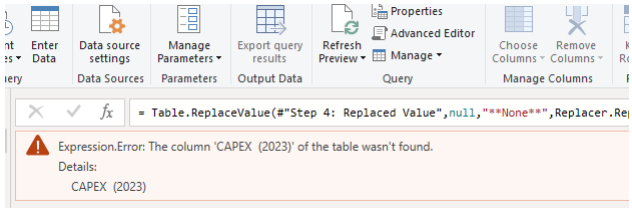

Fig. 6. Result of renaming a column in the source.

```
let
    Source = Excel.Workbook(File.Contents("C:\Users\ghaer\Downloads\Source.xlsx"), null, true),
    #"Sheet 1_Sheet" = Source{[Item="Sheet 1",Kind="Sheet"]}[Data],
    #"Step 2: Promoted Headers" = Table.PromoteHeaders(#"Sheet 1_Sheet", [PromoteAllScalars=true]),
    #"Step 3: Changed Type" = Table.TransformColumnTypes(#"Step 2: Promoted Headers",{{"ID", type text}, {"Name", type text}, {"Business Criticality", type text}, {"CAPEX  (2023)", Int64.Type}, {"Lifecycle: Active", type date}, {"Completion", type number}}),
    #"Step 3: Changed Type1" = Table.TransformColumnTypes(#"Step 3: Changed Type",{{"ID", type text}, {"Name", type text}, {"Business Criticality", type text}, {"CAPEX  (2023)", type text}, {"Lifecycle: Active", type text}, {"Completion", type text}}),
    #"Step 4: Replaced Value" = Table.ReplaceValue(#"Step 3: Changed Type1","","**None**",Replacer.ReplaceValue,{"ID", "Name", "Business Criticality", "CAPEX  (2023)", "Lifecycle: Active", "Completion"}),
    #"Step 5: Replaced Value1" = Table.ReplaceValue(#"Step 4: Replaced Value",null,"**None**",Replacer.ReplaceValue,{"ID", "Name", "Business Criticality", "CAPEX  (2023)", "Lifecycle: Active", "Completion"})
in
    #"Step 5: Replaced Value1"
```

Fig. 7. Generated M code of the transformation.

making Power Query a low-code platform, a contradictio in terminis.

B. Evolvability Issues in Power BI Desktop

When Power BI Desktop starts, it first makes the data transformed by Power Query available as tables. This data can be visualized on the dashboard canvas by selecting visual widgets and configuring them to represent the required data. The low-code aspect of Power BI Desktop lies in the configurability of the visuals, their filtering, and interactions. Via Filters, the data linked to the visuals is sliced. Via interactions, the data made available for visualisation on a visual can be influenced by the selection of data on another visual.

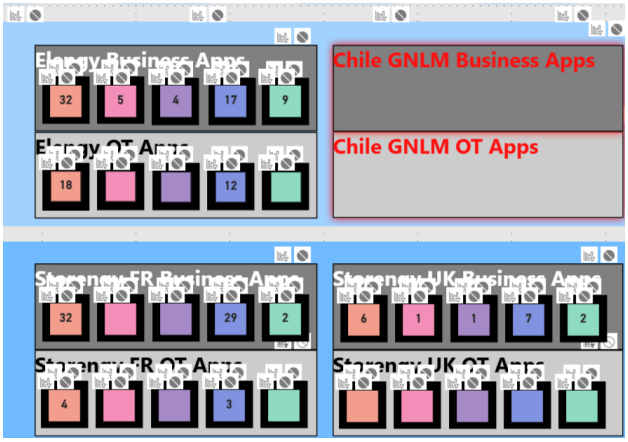

Fig. 8. Configuring Interactions: the small white/gray rectangles need to be clicked on for each activated or deactivated interaction with a newly added visual.

The interactions make the dashboard interactive and dynamic. By default, all visuals put on the modelling canvas interact with all existing visuals on the canvas. And herein lies the evolvability issues. It is not always desirable for visuals to interact. If you do not want selections on a visual to influence the other visuals on the canvas, you will have to disable the interactions on all other visuals manually. If you have ten visuals on your canvas, adding an eleventh means you need to disable interaction between the new and the existing visuals. The more visuals (i.e., the size of the system), the more effort is required to add a non-interacting visual. This is a clear example of a CE. The more visuals on the canvas, the more effort it takes. Figure 8 shows an example of a dashboard with over 100 visuals on it. Adding extra information to this visual is a tedious, error-prone task that can lead to unexpected dashboard behaviour and misinterpretation of data, the opposite of what a dashboard should be.

The root cause of this CE is a violation of the SoC. The introduction of a visual on a canvas should not be combined with configuring its interaction with existing visuals. Also, the default behaviour - interaction ON with existing visuals - results in the need for configuration. It would have been more logical to set this behaviour to default OFF and perform work proportional to the type of change: the need to interact with seven out of a hundred visuals requires an effort of seven, not a hundred.

V. Discussion

In this section, we consider the implications of the demonstrations' results from the previous Section. We begin with a framing of the demonstration and results, then discuss Power Query, followed by Power BI Desktop.

A. Framing of the Demonstration

It is important to frame the demonstration appropriately. The scenario we described assumes that Power BI is used for ETL and data visualization. There are, of course, scenarios in which a connection to a data source is made without any transformations occurring because the data is cleansed and transformed via other tools, and the end user connects only to the end result. But Power BI can do both, and over 30 years of personal experience in IT tooling leads us to believe that end users tend to exploit personal productivity tooling to the fullest, even if this results in highly unsustainable solutions.

This opinion warrants investigation using more rigorous methods. Additional research could be done on how frequently the type of issue we demonstrated appears in operational Power BI Dashboards. From the population of Power BI Dashboards used at the author's company, a sample could be drawn. In each sample, a change could be made to the data sources, similar to the one used in this paper: add a column to a source file and rename a column. The number of times this would result in a data load error would indicate how often the Power BI Dashboard creator has relied solely on low-code features to configure the ETL.

B. Implications of NS noncompliance in Power Query

When you do not know how to customize the M-code in Power Query, and thus enhance/replace what Power Query generates/expands, your ETLs will fail when there are changes in the source data. As you have no control over the source data, this impact is inevitable.

Microsoft may have intended Power BI as a tool for analyzing and reporting data, but the result is unsustainable if only the low-code/no-code features are used. We argue that such tools, seemingly designed for non-IT personnel, can

create tension between Business and IT. The tool is put in the hands of non-IT people, who will know better than IT how to interpret the data, but they are unaware of the tool's non-sustainability. When work is shared via publication and issues become visible to a large audience, IT is typically asked to resolve them, leading to rework and additional coding that cost more than initially expected and potentially contributing to Business-IT alignment tensions. The author (working for a multinational utilities company with over 98,000 employees) has witnessed such cases many times but recognises that this "anecdotal evidence" warrants further investigation. The research we propose would examine whether democratising IT capabilities (such as ETL and Data Visualisation) helps or adds tension to Business-IT alignment. After all, these tools are also socio-technical by nature [13].

In Section IV-A, we already touched on possible solutions. Why has Microsoft decided not to use the M language capabilities to expand low-code configuration into evolving code? The language is sufficiently rich to comply with AvT and DvT. A chain of ETL steps can be made SoS-compliant by implementing proper error handling, and because each transformation step addresses only one concern, SoC is also achievable. However, these features are available only to seasoned M-coders, not to low-code users.

C. Implications of NS noncompliance in Power BI Desktop

In Section IV-B, we saw that the default ON for interactions between visuals introduces a CE. Again, we ask why Microsoft made this design decision. In Power Query, the issues could still be fixed via M coding; in Power BI Desktop, there is no equivalent to steer the behaviour of the visualization widgets. Power BI Desktop provides the DAX language, but it is used to further process, filter, and extract information from the data. It does not allow control over the visualization widget's behaviour. There are other data visualization tools available, such as Graphana, in which the configuration of a visualization widget is stored in a JSON file. This opens the door for visualization widget expansion. In Power BI, this is not possible. We also suggest additional research in this area: which tools support expansion-based visualization widgets, and what should the structure of such a widget be to exhibit evolvability with respect to the addition/removal of widgets on a dashboard?

VI. Conclusion

The overarching argument we present in this paper is that using Power BI's low-code features yields a non-evolvable solution. We began by situating the topic in Section I, followed by an introduction to Power BI in Section II and to NS in Section III, as NS is the tool we use to evaluate evolvability. By means of demonstrating evolvability issues in Section IV, we present our arguments in Section V and point toward additional research.

References

[1] Apps run the world, [Online], Available: https://www.appsruntheworld.com/top-10-analytics-and-bi-software-vendors-and-market-forecast, [retrieved: March, 2026]

[2] Mordor Intelligence, [Online], Available: https://www.mordorintelligence.com/industry-reports/data-visualization-market, [retrieved: March, 2026]

[3] Grand View Research, [Online], Available: https://www.grandviewresearch.com/industry-analysis/data-visualization-tools-market-report, [retrieved: March, 2026]

[4] A. Ganeshan, E. Macari, J.O'Brien, K. Schlegel, and C. Long, "Magic Quadrant for Analytics and Business Intelligence Platforms," Gartner, IDG00822218, June 2025.

[5] A. Ganeshan, E. Macari, J.O'Brien, K. Schlegel, and C. Long, "Critical Capabilities for Analytics and Business Intelligence Platforms," Gartner, IDG00822221, June 2025

[6] Microsoft Learn, [Online], Available: http://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview, [retrieved: March, 2026]

[7] C. Ravi,"ETL (Extract, Transform & Load) Automation," International Journal of Emerging Trends in Computer Science and Information Technology, Volume 6, Issue 1, pp. 51-54, 2025

[8] T. Pellekoorne, "Building a Low-Code Platform for versatile Data Integration," Diss. Master's thesis, Technische Hochschule Mittelhessen (University of Applied Science), 2024

[9] H. Mannaert, J. Verelst, and P. De Bruyn, "Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design," ISBN 978-90-77160-09-1, Koppa, 2016.

[10] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," Science of Computer Programming, Volume 76, Issue 12, pp. 1210-1222, 2011.

[11] G. Haerens and H. Mannaert, "On the operationalization of composable architecture by means of NS theory," In PATTERNS 2025: The Seventeenth International Conference on Pervasive Patterns and Applications, pp 23-30, 2025.

[12] Microsoft Community, [Online], Available: https://community.fabric.microsoft.com/t5/Desktop/How-to-Add-a-New-Column-to-an-Existing-Power-Query-Without/td-p/4104466, [retrieved: March, 2026]

[13] N. Prinz, C. Rentrop, and M. Huber. "Low-Code Development Platforms-A Literature Review," AMCIS, 2021

A Methodology for Investigating AI Patterns Prevalence in Software Repositories

Srinath Perera[1], Hasinthaka Piyumal[1], Frank Leymann[2], and Rania Khalaf[1]
WSO2[1], University of Stuttgart[2]
Santa Clara, CA, USA[1], WSO2, Stuttgart, Germany[2]
e-mail: {srinath, hasinthaka, rania}@wso2.com [1], frank.leymann@iaas.uni-stuttgart.de[2]

***Abstract*—As Artificial Intelligence(AI)-based applications take off, a clear understanding of AI patterns can uplift the quality of AI applications. Many AI patterns have been proposed in the literature; however, their prevalence in real-life code has not yet been validated. Understanding the actual use of those patterns in practice can clarify our understanding both of the significance of these patterns and their utility. In this paper, we present a methodology to a) identify relevant patterns by mining the literature and then to b) validate their presence and prevalence in actual code repositories using active learning. To that end, we identify 14 AI pattern classes by mining 44 published AI pattern-related sources. Then we use an active learning approach to determine the prevalence of the most common pattern class across 100 GitHub open AI repositories. Using prevalence estimation, we propose bounds on the accuracy of the occurrences. The model achieves 56% accuracy and 55% recall in an 8-way classification task, significantly outperforming the 11% random-chance baseline. Furthermore, the prevalence estimation offers usable bounds for analyzing pattern applications. This methodology provides a robust foundation to start understanding how AI patterns are used in practice, a field that currently lacks empirical data.**

Keywords-patterns;pattern analysis; software; software engineering

I. INTRODUCTION

Large Language Models (LLMs) [1] now provide general-purpose, high-quality Artificial Intelligence (AI) capabilities that require little to no user-provided training data. These models have unlocked many previously infeasible use cases. Several patterns and abstractions including Retrieval-Augmented Generation (RAG) [2], ReAct [3], and agent-based frameworks [4] now help developers build AI applications. Software Design patterns [5] document recurring problems and solution templates, thereby improving software quality by enabling, communicating, and educating best practices.

As we will discuss in the related work section, many AI patterns have been proposed in the literature. Those patterns aim to capture the authors' observations and experiences. Validating those patterns by understanding their usage frequency in practice can clarify their relative importance, thereby improving the quality of AI-based applications. Furthermore, estimating usage frequency will help us verify candidate patterns using the rule of three (rule of X) [6].

We have identified 769 AI design pattern candidates proposed in the literature. Using word embeddings of those pattern descriptions and clustering, we have grouped them into 78 refined pattern candidates. Then, by manual inspection, we have categorized them into 14 pattern classes.

To verify the prevalence of pattern classes in practice, we selected 100 open-source GitHub repositories that implement real-world AI applications. From those repositories, we have extracted 2442 code communities. (A code community is a group of tightly coupled methods (procedures) in the call graph.)

Then, using an active learning-based approach [7], we built a classifier to detect the most common patterns listed above. Active learning builds a model with the help of an (e.g., human) oracle, aiming to minimize the number of data annotations (i.e., oracle queries). We proceed by starting with an initial model and iterative labeling of a few data-points that exhibit higher uncertainty in their predictions under the current model. Then, we rebuild the model with the new data and iterate. Consequently, we obtain an accuracy bound for how many times each of the above pattern classes will occur in the repositories.

The paper makes the following contributions.

- By combining methods active learning and human-in-the-loop annotations, we propose a robust methodology to detect patterns without pre-existing annotated datasets, thereby addressing a significant bottleneck in software engineering research: labeled data for niche or emerging patterns is rare.
- We propose a new chunking method to generate code embeddings useful for pattern detection by applying the Louvain method [**54**] to call graphs to identify "code communities" that serve as chunks.
- We propose applying prevalence estimation techniques (matrix inversion and Monte Carlo simulations) to estimate the true frequency of patterns with useful error bounds.
- We synthesized 769 pattern candidates into a refined taxonomy of 14 pattern classes. We provided one of the first empirical validations of AI pattern prevalence in real-life code by analyzing 100 open-source repositories.

The remainder of the paper is organized as follows. The section II discusses related work, and the following Section III describes the proposed methodology. Section IV describes the implementation details, followed by Section V, which describes the results. The section VI discusses the findings and lessons learned, and Section VII concludes the paper.

II. RELATED WORKS

This section discusses previous work on AI patterns and techniques for Mining Patterns from Code.

A. AI Patterns

Researchers and practitioners have proposed many AI patterns. Among post-LLM patterns, Huang [8] is a book on LLM design patterns. Gullí [9] and Liu et al. [10] discuss agent design patterns. Subramaniam [11] and Jain [12] provide the

practitioner's perspective. The AWS web page [13] provides a detailed description of best practices for implementing AI and ML applications on AWS. Also, Databricks documentation [14] lists a collection of Agent system design patterns. Furthermore, Arslan et al. [**32**] and Singh et al. [15] provide detailed surveys of RAG patterns; articles [16] and [11] present practitioners' perspectives on the same pattern.

Among pre-LLM patterns, Lakshmanan et al. [17] is a textbook on design patterns from the ML domain. Nalchigar [**53**] provides a detailed taxonomy of solution patterns for ML tasks such as clustering, classification, and anomaly detection. Washizaki [18] lists 15 design patterns (derived from a literature study) for building ML applications. Rodaz et al. [**40**] propose a mathematical formalism that sketches tasks around ML, but it is not digestible by developers. The Azure website [19] lists best practices for MLOps.

You can find a comprehensive list of sources we used to mine patterns from [20]. These include the sources mentioned above, as well as additional practitioner content, such as articles, blogs, and documentation.

B. Mining Pattern from Code

When detecting patterns in code, we need to first find a code representation and a pattern representation, and then match them. Calculating code representations involves a trade-off between losing information and maintaining generalizability. Pattern representations closer to code are often suboptimal as they fail to generalize across multiple languages. On the other hand, manually crafted pattern representations require significant effort and scale poorly to large numbers of patterns. Many techniques have been proposed for finding design patterns from code.

Search-based approaches use representations closer to code. Work such as Kramer et al. [21] and Dabain et al. [22] focused on detecting patterns using rules that describe them. These rules detect relationships between classes (e.g., inheritance hierarchies and method invocations). To reduce the overhead of pattern specification, Ghulam et al. [23] break GOF [5] patterns into 44 reusable atomic patterns, detect each using SQL, Regular Expressions, or AST parsing, and combine them to discover the patterns. Zdun et al. [24] turn microservice patterns into formal rules and provide a tool that automatically evaluates code in terms of compliance with these patterns.

A logical extension of this method is to represent patterns as subgraphs and to search the code graph for them. Mayvan [25] is an example of this approach. On the other hand, Tsantalis [26] used similarity scores between the code graph and the pattern subgraph, enabling more flexible matching. GEML [27] evolves a set of human-readable if-then-else rules using a greedy algorithm based on inexact graph matching to detect a pattern.

More recent work often focuses on ML-based statistical methods, which can avoid challenges of specifying patterns using rules or templates. Instead of specifying the pattern, they learn a pattern-identifier from sample data.

Among them, classification techniques such as those of Uchiyama et al. [28] and Dwivedi et al. [29] trained a classifier on software metrics such as Depth of Inheritance, Coupling, the number of methods, etc. Chihada et al. [30] use features extracted from an association graph to train the classifier. Zanoni et al. [31] use a combination of graph matching and ML techniques.

The downsides of initial ML techniques include the need for feature engineering, loss of information in features, and limited availability of training data.

Najam et al. [32] build a representation of the source code, apply the word2vec algorithm to construct an embedding space, and then use a classification algorithm. Pandey et al [19] use code embeddings (RoBERTa) generated from code with a KNN algorithm to achieve an F1 score of 0.91. This approach is very flexible as it can be used with minimal effort to detect new patterns if labels for the data are available.

Most of the above approaches work with benchmarks P-MART [33] and DPB [34], which primarily focus on software engineering patterns (e.g., GOF). No suitable dataset is available for detecting AI patterns.

Furthermore, a common limitation is that most benchmarks focus solely on GOF patterns. One notable exception is Fernández et al. [35], which first builds quantum circuits and then uses state-machine-based pattern-detection methods.

Our work is motivated by the results of Pandey et al. [19], which demonstrated the viability of embedding-based approaches. By retaining most of the information in code, embeddings avoid the need for feature engineering. However, embeddings are highly sensitive to the chunking method (i.e., which code is grouped together for calculating embeddings). We extended the work of Pandey et al. [19], which uses P-MART with clean code segments each including a single pattern, to detect patterns in real-life code repositories. Also, our work extends beyond well-known patterns (e.g., GOF) and limited annotated datasets by addressing the lack of training data via an active learning approach.

III. METHODOLOGY

Figure 1 depicts the methodology we used to estimate the occurrence of AI patterns in real-world AI applications as a flow chart. It has four steps. Each of the following subsections discusses four steps of the methodology in detail.

A. Step 1: Extracting pattern candidates from the literature

For mining AI patterns, we used the sources listed in the related work and practitioner articles, blogs, and documentation. You can find the detailed list from [20]. For each source, we extracted text, and for videos, we used the transcript.

We query an LLM (using Prompts [20]) to extract pattern candidates. We receive resulting pattern candidates in the template specified in the prompt.

To detect similarity between pattern candidates, we generated word embeddings for pattern description and clustered embeddings using the DBSCAN algorithm. We explored clustering algorithms DBSCAN, HDBSCAN, OPTICS, Bayesian

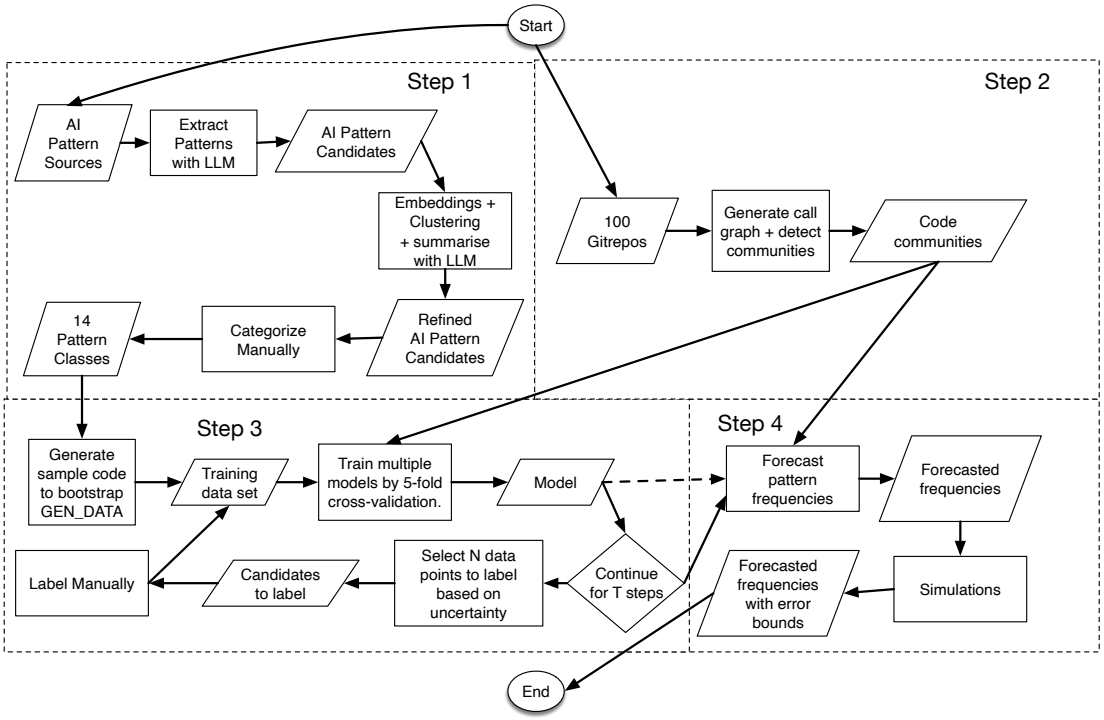

Figure 1. Proposed Methodology

Gaussian Mixtures, and K-Means. We picked DBSCAN by evaluating the even distribution of cluster sizes via silhouette score. We summarize each resulting cluster using an LLM with the prompt [20] and use them as refined pattern candidates.

Then, by manually inspecting pattern candidates, we categorize them under 14 pattern classes using the following criteria.

- We started with pattern classes: RAG, "Advanced LLM Prompting," "Forecasting with Classical Models," and "Multi-Agent Architecture," which were identified in related work.
- We tried to identify a pattern class for each refined-pattern-candidate that we identified by clustering. If we failed, we created a new pattern class. Following this approach, we added pattern classes "Evaluating LLM Results," "Multi-modal Data Processing and Prompting," "Using Tools with LLMs," "LLM-based User Intent Extraction," "LLM Fine Tuning," "Training & Alignment," "Enabling Reliability, Explainability, or Robustness," "LLM-based Planning, XoT, ReAct, or Reasoning," and "MLOps" using this approach.
- We also added 2 more pattern classes: "Preprocessing Text and Numerical Data" and "Model Abstraction," while manually labeling data in step four.

The classification of patterns and their refined candidates can be found in [20].

B. Step 2: Collecting Real-world Code Communities

A code community is a group of tightly coupled methods (procedures) in the call graph. To extract code communities, we curated a dataset of 100 repositories hosted on GitHub. The selection process filtered for Python repositories that possess between 250 and 1,000 stars. We manually verified each candidate to ensure it was a valid AI-related project. The complete list of selected repositories is provided in [20].

Manual inspection of 50 repositories with over 1,000 stars revealed that the majority were foundational frameworks, such as TensorFlow. Consequently, we targeted the 250–1,000 star range to intentionally isolate the 'application layer' of AI software, to understand how typical developers use AI in practice.

Embeddings will only work if chunks include full patterns and are reasonably small. For each repository, we generated a procedure call graph and detected communities using the Louvain method [**54**]. Since all functions within a single community are closely coupled, code within each community is likely to be related. Operating under the hypothesis that such code communities contain complete patterns, we treated all code in each code community as a single chunk and generated code embeddings for each community using `gemini-embedding-001`. We call this dataset $D_{unlabeled}$.

C. Step 3: Build a model with Active learning

We built a model to detect known code pattern classes by first creating a bootstrap version, selecting data for manual labeling via active learning criteria, and then iteratively rebuilding the model using five-fold cross-validation [36]. This cross-validation approach allows us to fully utilize all labeled data points.

To bootstrap the model, we prompted the LLM to generate multiple samples for pattern classes identified in step 1, which we will call D_{init}.

Input : D_{init} (Initial generated data), $D_{unlabeled}$ (Unlabeled embeddings), T (Total iterations)
Output: M_{final} (Trained pattern identifier model)

1 $V_{orig} \leftarrow \emptyset$ // Initialize original verified pool
2 $V \leftarrow D_{init}$ // Initialize current working pool
3 $M_{list} \leftarrow$ None
4 for $i \leftarrow 1$ to T do
5 $V \leftarrow$ handle_rare_classes(V)
// Step 1: Train model using current pool
6 $M_{list} \leftarrow$ train_hybrid_model_cv(V)
// Step 2: Active Learning - Identify uncertain samples
7 $S_{hl} \leftarrow$ find_uncertain_topN($M_{list}, D_{unlabeled}$)
// Step 3: Human-in-the-loop labeling
8 $D_{new} \leftarrow$ label_manually(S_{hl})
// Step 4: Update pools
9 $V_{orig} \leftarrow V_{orig} \cup D_{new}$
10 $D_{unlabeled} \leftarrow D_{unlabeled} - S_{hl}$
11 $V \leftarrow V_{orig}$
12 return M_{list}

Figure 2. Active Learning based Pattern Identifier

// Pseudocode for train_hybrid_model_cv(..)
Input: Verified Data $\mathcal{V}$
Output: Models M, F1 scores of models F
1 $\mathcal{N} \leftarrow$ {"lr", "knn", "svc"}
2 $C_{folds} \leftarrow \emptyset$ // fold configs
3 $\mathcal{F} \leftarrow \emptyset$
4 $O_{folds} \leftarrow \emptyset$
5 for $f \leftarrow$ split_to_folds($\mathcal{V}$) do
6 $D_{train} \leftarrow$ f.train
7 $D_{val} \leftarrow$ f.validation
8 $D_{test} \leftarrow$ f.test
9 $M_{list}, F_1 \leftarrow$ train($\mathcal{N}, D_{train}, D_{val}, \emptyset$)
// use algorithm in Figure 4 to forecast
10 $O_{folds} \leftarrow O_{folds} \cup$ forecast(M_{list}, D_{test})
11 $C_{folds} \leftarrow$ get_configs(M_{list})
12 $\mathcal{F} \leftarrow \mathcal{F} \cup \{F_1\}$
13 end
14 print_results(O_{folds}, D_{test})
// Aggregate results and train final ensemble
15 $M_{list}, \_ \leftarrow$ train($\mathcal{N}, V, \emptyset, C_{folds}$)
16 return $\{M_{list}, \mathcal{F}\}$

Figure 3. Building a model with Cross-validation

// Pseudocode for forecast(...)
Input: x Input data set, Models M, F1 scores of models F
Output: Y The final predicted class labels
/* Definitions: */
1 $\mathcal{C}$: The set of all possible class labels
2 $f_i \in F$: The F_1 score used as the raw weight for the i^{th} model
3 $P(y = c \mid m_i, x)$: The probability that model m_i assigns to class c for input x
// Ensemble Forecast Calculation
4 $Y = \underset{c \in \mathcal{C}}{\text{argmax}} \sum_{i=1}^{|M|} \left(\frac{f_i}{\sum_{j=1}^{n} f_j} \cdot P(y = c \mid m_i, x) \right)$
5 return Y

Figure 4. Final Forecast

The algorithm in Figure 2 explains the active learning based approach. At each step (algorithm 2, line 5), we dropped classes with fewer than 5% of the data and 20 data points. We do not know the true labels; hence, we cannot control the distribution of labeled data without significant effort. In the tradeoff between stable results vs. forecasting more classes, we chose stability. As steps progressed and more data became available, more classes will become available.

In `find_uncertain_topN()` function, we select data points to label from $D_{unlabeled}$ by forecasting them using current models, normalizing class probabilities of different models, averaging them to create overall predicted class probabilities, and performing margin sampling. Margin is the difference between the highest and second-highest classes in the forecast, and we select data points to label based on the smallest margin.

Algorithm in Figure 3 presents the logic for building a cross-validated model. And algorithm 4 describes how to perform the final forecast as a weighted average of probability distributions, with weights based on F1 scores across folds.

D. Step 4: Pattern Occurrence Estimation based on the results

Let m denote the model trained in step 3 to detect pattern classes in a code community. We utilize the normalized confusion matrix derived from the model's cross-validation, denoted as C. Given the set of code communities D (constructed in step 2), we define O_D as the observed distribution of pattern classes in D as predicted by m.

Assume T_D is the estimated true distribution of class in D.

$$\mathbf{O}_{samp} = \mathbf{C} \cdot \mathbf{T}_D \implies \mathbf{T}_D = \mathbf{C}^{-1}\mathbf{O}_{samp} \quad (1)$$

Given O_D, we can find T_D by using a linear solver.

When estimating T_D using our model, we face two sources of uncertainty: model misclassifications and sampling errors from O_D. We minimize the first error using equation 1, and the second by sampling from a multinomial distribution to estimate the true frequencies, then performing a Monte Carlo simulation to obtain error bounds. Algorithm 4 depicts our approach.

IV. Implementation Details

At every step, we use Gemini 3 Flash as the LLM. All experiments are done in an Intel Core™ i7-10510U × 8 processor with 16.0 GiB running Ubuntu 24.04.3 LTS

We already discussed the choice of clustering algorithm used in step 1 in the methodology. Furthermore, we tried dimension reduction with PCA and UMAP. We received the best results with DBSCAN with UMAP.

In step 3, for code embedding, we experiment with embedding methods Jina Embeddings v2 (Code), CodeSage Large v2, CodeRankEmbed, RoBERTa, CodeBERT, Voyage Code-2, Gemini Text Embedding 004, and selected Gemini Text

Input: C (Confusion Matrix), $\mathbf{O}$ (Observed counts), α (significance level), N (Iterations)
Output: $[L, U]$ (Confidence interval bounds)

$\mathcal{R} \leftarrow \emptyset$ // Initialize results collection
$C_{norm} \leftarrow$ normalize(C) // Normalize columns to sum to 1
$n \leftarrow \sum \mathbf{O}$ // Total sample count
$\mathbf{p} \leftarrow \mathbf{O}/n$ // Observed frequencies
for $i \leftarrow 1$ to N do
  // Sample from multinomial to simulate observation noise
  $\mathbf{O}_{samp} \leftarrow$ sample_multinomial(n, p)
  // Solve linear system using Equation 1
  $\mathbf{E} \leftarrow$ solve($C_{norm}, \mathbf{O}_{samp}$)
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\text{clip}(E)\}$ // Store clipped estimates
$L \leftarrow$ percentile($\mathcal{R}, \alpha/2$)
$U \leftarrow$ percentile($\mathcal{R}, 100 - \alpha/2$)
return $[L, U]$

Figure 5. Inversion-based Estimation via Monte Carlo Simulation

Embedding 00,4, which provided the best classification F1-score while building a bootstrapping model in active learning.

Unlike step 1, in step 3, using dimensionality reduction with the classifier model reduced the F1-score; Hence, we did not use any dimensionality reduction.

While building the model for step 3, we tried Logistic Regression (LR), Neural Networks, Support Vector Classifier (SVC), Naive Bayes, Random Forest, XGBoost, Gradient Boosting, K-Nearest Neighbour (KNN), and Decision Tree. We used LR, SVC, and KNN, which provided the best individual F1 scores in the final averaging ensemble.

While building the model in step 3, we achieved about a 4% improvement in F1 score using the averaging ensemble method in code listing 3. Building a linear regression stacked model using outputs from a neural network and a linear regression model also yielded good results. Still, we chose the averaging ensemble method because the stacked model is complex and highly susceptible to overfitting.

In step 3, we used the following workflow for manually labeling code communities. Given a code community (cc) and a predicted pattern class pc, we use the LLM to generate a code description for cc, ask the LLM to judge whether the forecasted pattern class pc is correct, and describe its decision. We then manually verified that the LLM's judgment was correct. If the prediction (pc) is wrong, we read the description of cc and propose an alternative pattern class. [20] lists Prompts used for this step.

During active learning, in the third iteration, we tried building the model both with and without LLM-generated sample data, and it performed better with only verified data. So we only used verified data from the third iteration onward.

In step 4, for the simulation, we assume the confusion matrix of the model developed in step 3 is representative. As discussed in the methodology, we model the sampling error as independent categorical draws from a multinomial distribution, with the observed category counts as the basis for the distribution. We used 20000 as the iteration count, increasing it until repeated tests did not change the outputs.

V. Results

Manually verified data had pattern frequencies as follows: Forecasting with Classical Models (64), None (64), LLM-based Multimodal Generative Prompting (52), Preprocessing Text and Numerical Data (51), Using Tools with LLMs (31), RAG (28), Agent Architecture (20), Model Abstraction(20), Evaluating LLM Results (18), MLOps (18), Advanced LLM Prompting (14), LLM-based Planning, XoT, ReAct, or Reasoning (12), Enabling Reliability, Explainability, or Robustness (8), LLM-based User Intent Extraction (8), and LLM Fine-Tuning, Training & Alignment (4).

However, the rest of the results will only have 7 pattern classes because the model will automatically map any class less than 5% and 20 instance to the None class. Table I shows the results after the fourth iteration.

The verified data column shows the number of manually labeled data points available. As expected, classes with fewer training samples have lower F1 scores.

Table 6 shows the confusion matrix.

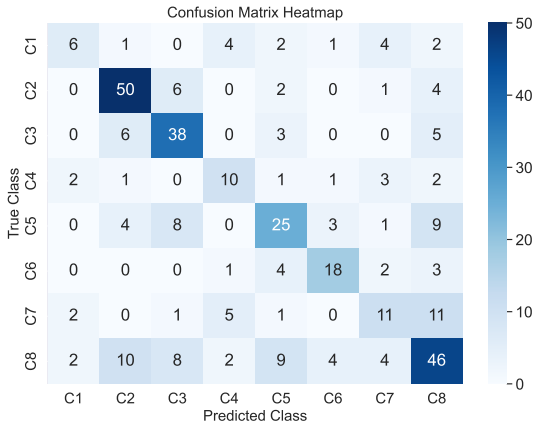

Figure 6. Weighted Vote Confusion Matrix

A strong main diagonal indicates the model's strength. As we discussed in the methodology, most classes are often misclassified as "None", and the "None" class often gets misclassified. Hence, it is a common source of error. Considering the often misclassified patterns, "Using tools with LLM" is often misclassified as "None", which could be because "tool use" is not very prominent in code and lacks a strong signal. Furthermore, classical models and preprocessing both get confused by "None" in both directions, which may be because these patterns include a wide variety of techniques that are harder to generalize with a small set of samples.

Figure 7 depicts our results from the prevalence estimation described in step 4.

There are several zeros in bounds for the last two classes, which could occur when a class is affected by high false-positive noise from other classes or when it is often misclassified as other classes. We conducted a sensitivity test for both

TABLE I. CLASSIFICATION PERFORMANCE METRICS ACROSS CATEGORIES.

	Category	Precision	Recall	F1-Score	Verified Data
C1	Agent Architecture	0.50	0.30	0.38	20
C2	Forecasting with Classical Models	0.69	0.79	0.74	63
C3	Multimodal Generative Prompting	0.62	0.73	0.67	52
C4	Model Abstraction	0.45	0.50	0.48	20
C5	Preprocessing Text and Numerical Data	0.53	0.50	0.52	50
C6	RAG	0.67	0.64	0.65	28
C7	Using Tools with LLMs	0.42	0.35	0.39	31
C8	None	0.56	0.54	0.55	85
	Overall Values	**0.56**	**0.55**	**0.55**	

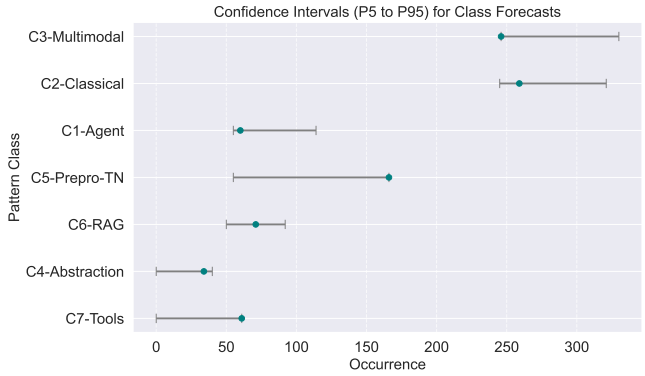

Figure 7. Estimated Pattern Class Prevalence with 95% confidence intervals

cases and found that the probable cause is that "Tool Use for LLMs" is often misclassified as "None" as shown in the confusion matrix in Figure 6.

Smaller training samples likely cause wider confidence intervals for the last two classes. When we exclude those, the confidence interval variation is 45%, which provides usable estimates for our problem. Furthermore, confidence levels indicate when estimates are unreliable, and users can address that by running more iterations of the algorithm. Since the algorithm selects labeling data based on uncertainty, there is a good chance that underrepresented classes will be labeled in subsequent iterations.

Where are the prevalence results at the current accuracy level useful? First, it can confirm the wider availability of 5/7 pattern classes. Furthermore, it can give us relative availability of different pattern classes. For example, even using given confidence levels, we can argue that pattern classes C2 and C3 are much more common than C1, C5, and C6. Such understanding can help us focus our attention on teaching these patterns. However, the current accuracy level is not enough to identify a specific pattern given a code segment, which is a future direction we plan to explore.

The following lists concrete examples organized by pattern class, found based on the model's forecast for extracted communities.

- Agent Architecture - multi-agent debate, multi-agent orchestration, communication through an agent bus, coordination through group chat.
- LLM-based Multimodal Generative Prompting - Predicting object centers, dimensions, and local offsets, inferring an object's 3D position, bounding box prediction, motion prediction, sequential detections of an object from Lidar data, resizing and padding 3D tensors, generating synthetic training data.
- Model Abstraction - abstracting multiple AI backends or local/ cloud producers, routing, adopting Prompts via template, and managing API keys
- Preprocessing Text and Numerical Data - anonymizes, binning, type conversions, data dictionary, extracting metadata, text normalization, categorical feature encoding
- Retrieval Augmented Generation(RAG) - search with vector database, using custom knowledge bases, caching, and retriever component

VI. DISCUSSION AND LESSONS LEARNED

One code community can house multiple patterns. However, to keep manual data labeling simple, we have only used a single label. However, our output provides a probability distribution across all classes, which can be used to detect multiple patterns.

Contrasting our results with those of Pandey et al. [19], which achieved an F1-score of 0.91, it is worth noting that we also tried the same algorithms and embeddings. The likely difference is the curated nature of PMART vs. the complexity of real-life code we used, which often includes additional logic. Furthermore, as we discussed in the results section, users of our methodology can assess the reliability of estimates using error bounds. If they require tighter error bounds, they can likely achieve them by running more iterations of the step three active learning algorithm, which likely yields more training data for those classes and thereby improves the error bounds. Given the limited understanding of design pattern frequency in the real world, even the bounds we reported after four iterations remain usable for 5 out of the 7 classes and represent a significant step forward.

In step 3, we folded classes that had fewer than 5% or 20 samples into the None class, which is a practical choice. Some patterns are common, while others are rare, and naturally, there are many potential patterns. To make our approach stable,

we had to draw the line somewhere. When the number of samples in a class is small, when combined with 5-fold cross-validation, the F1-score changed significantly (>10%) due to each misclassified sample, thus becoming unreliable [37].

However, we acknowledge that this practical choice reduces the granularity of results, as artificial non-class can interfere with other classes. Because the boundary between what is considered a pattern and what is not is fluid (e.g., subjective in how abstract we want the problem and solution to be), we believe our choice is still useful. If needed, practitioners can often get more granularity by labeling more data points.

We extend our work beyond basic software patterns (e.g., GOF) and limited annotated datasets by addressing the lack of training data via an active learning approach. The methodology we used is highly independent of AI patterns, and the same approach is likely to work well with other types of patterns. The ability to detect patterns by building a classifier on embeddings has been demonstrated by Pandey et al. [19] using the P-MART data set, and by this paper on real-world AI projects. It is likely that other domains also contain sufficient information in their embeddings.

Detecting patterns via embeddings is highly sensitive to the choice of chunking method. Initially, we generated embeddings per file in the repository, resulting in poor classification performance in step 3. We needed a chunking method that would keep all relevant code about a pattern in a single chunk. Given that goal, typical chunking approaches like fixed size, pooling, by function, or by class would not work. AST-like methods (e.g., CAS [**[66]**) also focus on code syntax structure not the dependacies. In contrast, a call graph naturally captures dependencies similar to code graph-based approaches used in related work(e.g., Mayvan [25], Tsantalis [26]). It is much more likely that code from the same pattern, which is more closely related, is more connected in the call graph. This is also a natural extension of Codegrag [38], which has used a similar idea for code retrieval. The performance improved significantly after adopting the call graph-based communities as chunks.

We tried to detect patterns by clustering the code communities collected in step 2 and then interpreting and summarizing the resulting clusters. This approach did not work well. However, it may be possible to fine-tune the embeddings using techniques such as contrastive learning improve the separation between clusters, which is also a future work we plan to explore.

Methodology uses LLM in three ways, and each can introduce bias. First, pattern candidates were extracted using the LLM and manually curated to identify pattern classes. Manual curation should help reduce bias, but will not address bias shared between LLMs and humans. Second, although we bootstrapped with LLM-generated data in the first round, after 3rd round, we dropped LLM-generated data because only using labeled data gave better results. Third, LLM-assisted manual labeling can also introduce bias. Exploring the use of LLM as a judge technique to reduce such bias is a useful direction for future research.

VII. Conclusion and Future Work

In this paper, we aim to enable and shift the exploration of AI design patterns toward empirical validation using software repositories. To that end, we address three challenges. Embedding-based pattern detection is highly susceptible to chunking methods. We proposed a new chunking method based on call graph-based community detection. The performance of the resulting model validates our method. Second, AI has limited annotated pattern datasets, which we address through an active learning-based approach. Third, we address both sampling and classification errors while estimating pattern prevalence by using a matrix-inversion-based technique and Monte Carlo simulations.

The model achieves 56% accuracy and 55% recall in an 8-way classification task, which is 5 times higher than the 11% random-chance baseline. Prevalence estimation provided useful error bounds for 5 of 7 classes. In other cases, users can detect unreliable estimates through error bounds and discard them, or obtain better estimates by running more iterations in step three.

We synthesized 769 pattern candidates into a refined taxonomy of 14 pattern classes. Most patterns show more than three examples required to pass the rule of three. Frequency estimates for those classes confirm our understanding in some cases (e.g., Preprocessing Text and Numerical Data and Classical Models) and open new avenues of inquiry in others (Multimodal prompting and RAG).

This work opens several avenues for future research. First, applying the techniques in other domains to explore the universality of the approach (e.g., applying the same techniques for microservices, quantum computing), second, studying the relationships between possible bounds vs. the number of training samples for each class, and third, exploring techniques, such as SLM fine-tuning and constructive learning, to improve model performance. All information required to recreate the above work, including code, labeled data points from step 4, and code, is available in [20].

References

[1] Y. Chang et al., "A survey on evaluation of large language models", *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024. DOI: 10.1145/3641289.

[2] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[3] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models", *arXiv preprint arXiv:2210.03629*, 2023.

[4] J. He, C. Treude, and D. Lo, "LLM-based multi-agent systems for software engineering: Literature review, vision, and the road ahead", *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–30, 2025. DOI: 10.1145/3702989.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[6] C. Kohls and S. Panke, "Is that true...? thoughts on the epistemology of patterns", in *Proceedings of the 16th Conference on Pattern Languages of Programs*, 2009, pp. 1–14.

[7] A. Tharwat and W. Schenck, "A survey on active learning: State-of-the-art, practical challenges and research directions", *Mathematics*, vol. 11, no. 4, p. 820, 2023.

[8] K. Huang, *LLM Design Patterns: A Practical Guide to Building Robust and Efficient AI Systems*. O'Reilly Media, 2025.

[9] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, "Agentic retrieval-augmented generation: A survey on agentic rag", *arXiv preprint arXiv:2501.09136*, 2025.

[10] A. Gullí, *Agentic Design Patterns*. Packt Publishing, 2024.

[11] B. Subramaniam, "Emerging patterns in building GenAI products", 2024, Accessed: Jan. 22, 2026. [Online]. Available: https://martinfowler.com/articles/gen-ai-patterns/.

[12] A. Jain, "Agentic AI architectures and design patterns", 2024, Accessed: Jan. 22, 2026. [Online]. Available: https://medium.com/@anil.jain.baba/agentic-ai-architectures-and-design-patterns-288ac589179a.

[13] Amazon Web Services, "AWS prescriptive guidance: Patterns: AI & machine learning", 2026, Accessed: Jan. 22, 2026. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/machinelearning-pattern-list.html.

[14] Databricks, "Agent system design patterns", 2026, Accessed: Jan. 22, 2026. [Online]. Available: https://docs.databricks.com/aws/en/generative-ai/guide/agent-system-design-patterns.

[15] J. Alammar, "Retrieval-augmented generation (RAG) patterns and best practices", InfoQ, 2024, Accessed: Jan. 22, 2026. [Online]. Available: https://www.youtube.com/watch?v=eUY9i1CWmUg.

[16] Neo4j, "GraphRAG field guide: RAG patterns", 2026, Accessed: Jan. 22, 2026. [Online]. Available: https://neo4j.com/blog/developer/graphrag-field-guide-rag-patterns/.

[17] V. Lakshmanan, S. Robinson, and M. Munn, *Machine Learning Design Patterns*. O'Reilly Media, Inc., 2020.

[18] H. Washizaki et al., "Software-engineering design patterns for machine learning applications", *Computer*, vol. 55, no. 3, pp. 30–39, 2022. DOI: 10.1109/MC.2021.3139049.

[19] S. K. Pandey et al., "Design pattern recognition: A study of large language models", *Empirical Software Engineering*, vol. 30, no. 3, p. 69, 2025.

[20] "Ai patterns github repository", 2026, Accessed: Jan. 27, 2026. [Online]. Available: https://github.com/wso2-incubator/ai-patterns.

[21] C. Kramer and L. Prechelt, "Design recovery by automated search for structural design patterns in object-oriented software", in *Proceedings of WCRE'96: 3rd Working Conference on Reverse Engineering*, IEEE, 1996, pp. 208–215. DOI: 10.1109/WCRE.1996.558906.

[22] H. Dabain, A. Manzer, and V. Tzerpos, "Design pattern detection using FINDER", in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1554–1560. DOI: 10.1145/2695664.2695755.

[23] G. Rasool and P. Mäder, "Flexible design pattern detection based on feature types", *Automated Software Engineering*, vol. 18, no. 3-4, pp. 339–365, 2011. DOI: 10.1007/s10515-011-0084-2.

[24] U. Zdun, E. Navarro, and F. Leymann, "Ensuring and assessing architecture conformance to microservice decomposition patterns", in *International Conference on Service-Oriented Computing*, Springer, 2017, pp. 411–429.

[25] B. B. Mayvan and A. Rasoolzadegan, "Design pattern detection based on the graph theory", *Knowledge-Based Systems*, vol. 120, pp. 211–225, 2017.

[26] 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. DOI: 10.1109/TSE.2006.112.

[27] 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, pp. 110–919, 2021.

[28] S. Uchiyama, H. Washizaki, and Y. Fukazawa, "Design pattern detection using software metrics and machine learning", in *First International Workshop on Model-Driven Software Migration (MDSM 2011)*, 2011, pp. 38–42.

[29] A. K. Dwivedi, A. Tirkey, and S. K. Rath, "Software design pattern mining using classification-based techniques", *Frontiers of Computer Science*, vol. 12, no. 5, pp. 908–922, 2018.

[30] A. Chihada, V. Arnaoudova, L. M. Eshkevari, G. Antoniol, and Y.-G. Gueheneuc, "Source code and design conformance, design pattern detection from source code by classification approach", *Applied Soft Computing*, vol. 26, pp. 357–367, 2015. DOI: 10.1016/j.asoc.2014.09.043.

[31] M. Zanoni, F. A. Fontana, and F. Stella, "On applying machine learning techniques for design pattern detection", *Journal of Systems and Software*, vol. 103, pp. 102–117, 2015. DOI: 10.1016/j.jss.2015.01.037.

[32] N. Nazar, A. Aleti, and Y. Zheng, "Feature-based software design pattern detection", *Journal of Systems and Software*, vol. 185, pp. 111–179, 2022.

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

[34] F. A. Fontana, A. Caracciolo, and M. Zanoni, "DPB: A benchmark for design pattern detection tools", in *2012 16th European Conference on Software Maintenance and Reengineering*, IEEE, 2012, pp. 235–244. DOI: 10.1109/CSMR.2012.33.

[35] M. Fernández-Osuna, M. A. Pérez-Delgado, M. Rojo-Martínez, and M. Piattini, "Exploring design patterns in quantum software: A case study", *Computing*, vol. 107, no. 5, pp. 1–31, 2025. DOI: 10.1007/s00607-024-01365-z.

[36] R. D. King, O. I. Orhobor, and C. C. Taylor, "Cross-validation is safe to use", *Nature Machine Intelligence*, vol. 3, no. 4, pp. 276–276, 2021.

[37] C. Beleites, U. Neugebauer, T. Bocklitz, C. Krafft, and J. Popp, "Sample size planning for classification models", *Analytica chimica acta*, vol. 760, pp. 25–33, 2013.

[38] K. Du et al., "Codegrag: Bridging the gap between natural language and programming language via graphical retrieval augmented generation", *arXiv preprint arXiv:2405.02355*, 2024.