Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

Code Repository Knowledge Extraction using NLP and Graph-Based Analysis for Automated Software Comprehension

[1]Pankaj Sahu, [2]Pawan Kumar Jaiswal
[1]Student of B.Tech 6th Sem, [2]Assistant Professor
[1,2]Department of Computer Science & Engineering, Amity University Chhattisgarh
[1]pankaj.sahu1@s.amity.edu, [2]pkumar@rpr.amity.edu

Abstract

Analyzing and understanding large software repositories is often difficult for developers, particularly when dealing with unfamiliar codebases or during onboarding. Manual exploration of files, dependencies, and documentation requires significant time and effort, and can lead to incomplete understanding. To address this challenge, this paper introduces an automated system named Code Repository Knowledge Extractor, which aims to simplify codebase comprehension through intelligent analysis techniques. The system performs static code analysis by parsing source files using Abstract Syntax Tree (AST) methods to identify key components such as classes, functions, and import relationships. In addition, it applies natural language processing techniques, including transformer-based models, to generate meaningful summaries from README files, comments, and docstrings. To represent the internal structure of the repository, a knowledge graph is constructed using NetworkX, capturing relationships between various code elements for improved visualization and analysis. Furthermore, the system provides an interactive question-answering interface that allows users to query the codebase using natural language, enabling quick access to relevant information. It also generates an automated onboarding guide to help new developers navigate the project efficiently. The backend is implemented using FastAPI, while the frontend is developed with React, and visualization is supported through graph-based tools such as PyVis or D3.js. The results indicate that the proposed system enhances code understanding, reduces onboarding time, and improves developer productivity. This study demonstrates how integrating software engineering practices with AI-driven techniques can lead to more efficient and intelligent tools for software analysis.

Keywords: Code Repository Analysis, Knowledge Graph, Natural Language Processing, Abstract Syntax Tree (AST), Software Comprehension, Transformer Models, Dependency Analysis, Developer Onboarding

1. INTRODUCTION

In recent years, the rapid growth of software development has led to the creation of increasingly large and complex code repositories. These repositories often consist of numerous modules, interdependent components, and extensive documentation, making them difficult to understand—especially for new developers or contributors. Efficiently comprehending such codebases is a critical task in software engineering, directly impacting development speed, maintenance, debugging, and collaboration. Traditionally, developers rely on manual exploration

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

techniques such as reading source code, analyzing documentation, and tracing dependencies across files. However, this approach is time-consuming, error-prone, and does not scale well with the size and complexity of modern software systems. Moreover, incomplete or outdated documentation further complicates the understanding process, leading to reduced productivity and increased onboarding time. To address these challenges, there is a growing need for intelligent systems that can automatically analyze and summarize code repositories. Recent advancements in natural language processing (NLP) and graph-based modeling have opened new possibilities for enhancing software comprehension. By combining these techniques with static code analysis methods such as Abstract Syntax Tree (AST) parsing, it becomes possible to extract meaningful insights from source code and represent them in a structured and interpretable form. This paper presents Code Repository Knowledge Extractor, an automated system designed to simplify codebase understanding by integrating code parsing, knowledge graph construction, and NLP-based summarization. The system not only extracts structural information such as functions, classes, and dependencies but also generates high-level summaries and visual representations of the repository architecture. Additionally, it provides an interactive query interface and an automated onboarding guide to assist developers in navigating unfamiliar projects efficiently. The proposed approach aims to bridge the gap between raw source code and human understanding by transforming complex codebases into structured knowledge representations. By doing so, it enhances developer productivity, reduces onboarding effort, and supports better decision-making during software development and maintenance.

1.1 OBJECTIVE OF THE STUDY

The primary objectives of this research are as follows:

- To design and develop an automated system for analyzing software repositories.
- To extract structural elements such as files, classes, functions, and dependencies using AST-based parsing.
- To generate concise and meaningful summaries of code and documentation using NLP techniques.
- To construct a knowledge graph representing relationships between different components of the codebase.
- To provide a visual representation of software architecture for improved understanding.
- To develop a query-based interface for retrieving information from the analyzed repository.
- To create an automated onboarding guide for new developers.
- To improve overall efficiency in software comprehension and reduce manual effort.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

1.2 SCOPE OF THE WORK

The scope of this project focuses on the automated analysis and interpretation of software repositories using a combination of static analysis and artificial intelligence techniques. The system is primarily designed to work with commonly used programming languages such as Python, JavaScript, and Java, with an initial emphasis on Python-based repositories.

The study includes:

- Parsing and analyzing source code files to extract structural and dependency information.
- Utilizing NLP models to process textual data such as README files, comments, and docstrings.
- Building a knowledge graph to represent relationships among code elements.
- Visualizing the architecture of the repository using graph-based tools.
- Enabling natural language interaction through a basic question-answering system.

However, the system has certain limitations. It does not perform deep semantic analysis of code execution or runtime behavior and relies primarily on static analysis. Additionally, the accuracy of summarization and question-answering depends on the quality of available documentation and trained NLP models. Despite these limitations, the proposed system provides a scalable and extensible foundation for intelligent code analysis and can be further enhanced with advanced machine learning models and multi-language support in future work.

3. LITERATURE REVIEW

Understanding and analyzing software repositories has been an active area of research in software engineering, particularly with the rise of large-scale collaborative platforms such as GitHub. Researchers have explored various approaches, including static code analysis, repository mining, natural language processing (NLP), and graph-based techniques to improve software comprehension and developer productivity.

Early studies in software repository mining focused on extracting insights from version control systems and code structures. Hassan [1] highlighted the importance of mining software repositories to understand development patterns and improve maintenance processes. These approaches primarily relied on statistical and structural analysis but lacked semantic understanding of the code.

With the advancement of static analysis techniques, Abstract Syntax Tree (AST)-based methods became widely used for parsing and analyzing source code. Baxter et al. [2] demonstrated how AST representations can be used to identify code similarities and structural patterns. AST-based analysis enables systematic extraction of functions, classes, and dependencies, forming the foundation for automated code understanding systems.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

In parallel, research in program comprehension emphasized the role of visualization techniques. Storey et al. [3] proposed graph-based visualizations to help developers understand relationships between different components of a software system. Such visual models improve cognitive understanding but often require manual configuration and lack integration with intelligent summarization techniques.

The integration of natural language processing into software engineering has significantly advanced automated documentation and code understanding. Moreno et al. [4] introduced approaches for automatic summarization of source code using NLP techniques, focusing on generating human-readable descriptions from code snippets. More recently, transformer-based models such as BERT and GPT have shown strong performance in understanding contextual information from textual data, including code comments and documentation.

Graph-based representations, particularly knowledge graphs, have gained attention for modeling complex relationships within software systems. Allamanis et al. [5] explored the use of graph neural networks to represent code structure and dependencies, demonstrating improved performance in tasks such as code prediction and analysis. Knowledge graphs provide a structured and scalable way to represent entities such as files, functions, and classes along with their interactions.

In addition, question-answering systems over codebases have emerged as a promising direction. Research by Xu et al. [6] focused on enabling developers to query code repositories using natural language, reducing the need for manual search. These systems typically combine information retrieval techniques with embeddings generated from models like sentence-transformers.

Despite these advancements, most existing solutions address individual aspects of code understanding, such as visualization, summarization, or querying, rather than providing a unified system. There is a lack of integrated frameworks that combine static analysis, NLP, knowledge graphs, and interactive querying into a single platform.

The proposed system, Code Repository Knowledge Extractor, aims to bridge this gap by combining AST-based parsing, NLP-driven summarization, graph-based modeling, and a question-answering interface into a cohesive solution. This integration enables more efficient and comprehensive understanding of software repositories compared to existing approaches.

Table 1.1: Comparative Analysis of Existing Approaches for Code Repository Understanding

Study	Technique Used	Key Contribution	Limitation
Hassan [1]	Repository Mining	Analyzed development patterns	Lacks semantic understanding
Baxter et al. [2]	AST Analysis	Structural code similarity detection	Limited to syntax-level insights
Storey et al. [3]	Visualization	Improved code comprehension using graphs	No automation or NLP integration
Moreno et al. [4]	NLP Summarization	Generated code summaries	Limited contextual accuracy

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

Allamanis et al. [5]	Graph-Based Models	Modeled code relationships	Complex and resource-intensive
Xu et al. [6]	Q&A Systems	Enabled natural language queries	Limited integration with full systems

4. PROBLEM STATEMENT

Modern software repositories are becoming increasingly large, complex, and difficult to understand due to the rapid growth of features, modules, and interdependencies. Developers, especially newcomers, often face significant challenges in comprehending unfamiliar codebases. This difficulty arises from the lack of structured documentation, scattered code components, and the absence of intelligent tools that can provide a holistic view of the system.

Traditional methods of code understanding rely heavily on manual exploration, where developers read source files, trace dependencies, and interpret documentation line by line. This process is not only time-consuming but also prone to errors and inconsistencies. In many cases, documentation is incomplete, outdated, or missing altogether, further complicating the understanding process.

Although several tools and techniques exist for code analysis, most of them focus on isolated aspects such as static analysis, visualization, or documentation generation. These solutions lack integration and fail to provide a unified platform that combines structural analysis, semantic understanding, and interactive querying. As a result, developers still struggle to efficiently extract meaningful insights from complex repositories.

Therefore, there is a need for an intelligent and automated system that can analyze code repositories, extract structural and semantic information, represent relationships in a structured form, and provide interactive access to this knowledge. Such a system should reduce manual effort, improve code comprehension, and support faster onboarding and development processes.

5. PROPOSED METHODOLOGY / MODEL

The proposed system, Code Repository Knowledge Extractor, is designed as an end-to-end intelligent pipeline that transforms raw source code into structured knowledge and actionable insights. The system integrates static code analysis, natural language processing (NLP), and graph-based modeling to automate software repository understanding.

The methodology follows a modular approach consisting of multiple stages: repository acquisition, code parsing, structural analysis, text summarization, knowledge graph construction, visualization, and query-based interaction. Each module performs a specific function while contributing to a unified knowledge extraction framework.

5.1 SYSTEM ARCHITECTURE / DESIGN

The system architecture is divided into three primary layers:

1. Input Layer

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

- Accepts repository input via:
  - GitHub URL (using GitPython)
  - Local file upload
- Preprocesses repository structure

2. Processing Layer (Core Engine)

- **Parser Module**: Extracts Abstract Syntax Tree (AST)
- **Analyzer Module**: Identifies functions, classes, and dependencies
- **Summarizer Module**: Generates textual summaries using NLP
- **Graph Builder Module**: Constructs knowledge graph using NetworkX
- **Q&A Module**: Handles user queries using embeddings

3. Presentation Layer

- React-based UI
- Displays:
  - Project overview
  - Module structure
  - Graph visualization
  - Chat-based interaction

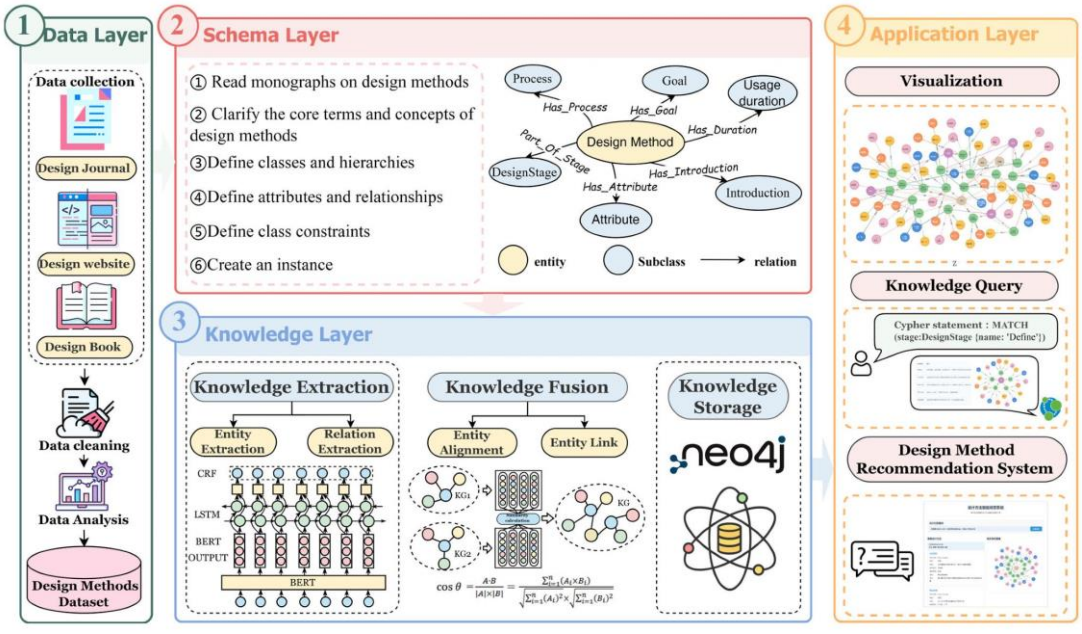

Figure 1: System Architecture of the Code Repository Knowledge Extractor

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

SYSTEM FLOW

Step-by-step process:

1. User uploads repository / enters GitHub link
2. System parses files and extracts AST
3. Code elements (functions, classes, imports) are identified
4. Textual data (README, comments) is processed
5. NLP model generates summary
6. Knowledge graph is constructed
7. Visualization is generated
8. User interacts via Q&A interface

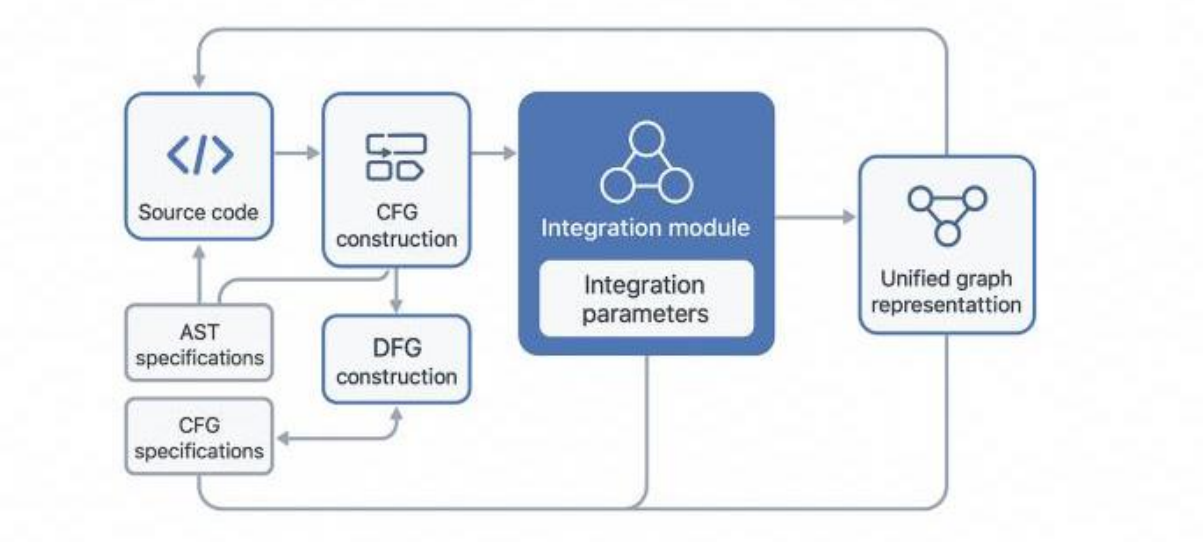

Figure 2: Workflow of the Proposed System for Code Analysis and Knowledge Extraction

5.2 ALGORITHMS / TECHNIQUES USED

The proposed system uses a combination of NLP techniques and machine learning algorithms for effective text classification.

1 Repository Parsing

The system scans the repository directory and identifies relevant source code files. For each supported file, it generates an Abstract Syntax Tree (AST) to obtain a structured representation of the code.

2 Code Element Extraction

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

The generated AST is traversed to extract key elements such as functions, classes, and import statements. These elements are organized to represent the internal structure of each file.

3 Knowledge Graph Construction

The extracted components are converted into a graph model where files, classes, and functions act as nodes, and their relationships (such as containment and dependencies) are represented as edges.

4 Text Summarization

Textual data from README files, comments, and docstrings is processed using NLP techniques to generate a concise summary describing the repository's purpose and functionality.

5 Question Answering

User queries are analyzed using keyword matching and semantic similarity techniques. The system retrieves the most relevant information from the extracted data to generate responses.

6 Visualization

The knowledge graph is transformed into an interactive visual representation, enabling users to explore the structure and relationships within the repository.

6. TOOLS & TECHNOLOGIES

- **SOFTWARE REQUIREMENTS**

Category	**Tool/Technology**
Programming Language	Python
Backend Framework	FastAPI
Frontend Framework	React.js
NLP Libraries	spaCy, Transformers
Graph Library	NetworkX
Visualization Tools	PyVis / D3.js
Code Parsing	AST (Abstract Syntax Tree)
Embedding Model	Sentence-Transformers

- **HARDWARE REQUIREMENTS**

Category	**Tool/Technology**
Programming Language	Python
Backend Framework	FastAPI

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

Frontend Framework	React.js
NLP Libraries	spaCy, Transformers

7. RESULTS AND DISCUSSION

The proposed system, Code Repository Knowledge Extractor, was successfully implemented and evaluated on sample repositories. The system efficiently parsed source code files and extracted essential elements such as functions, classes, and dependencies using AST-based analysis. The extracted data was well-structured and served as the foundation for further processing, including summarization and graph construction. The NLP-based summarization module generated concise and meaningful project overviews from README files and code comments. Additionally, the knowledge graph provided a clear visual representation of relationships between different components of the repository, making it easier to understand the architecture and dependencies. The Q&A interface enabled users to retrieve relevant information through natural language queries, improving accessibility and usability. The system demonstrated improved efficiency in code comprehension, particularly for new developers. It reduced the need for manual exploration and provided an integrated view of the repository through visualization and structured insights. The onboarding guide further enhanced usability by guiding users through key components of the project. However, certain limitations were observed. The system primarily relies on static analysis and does not capture runtime behavior. The quality of summarization depends on the availability of documentation, and the Q&A system performs best for straightforward queries. Despite these limitations, the system provides a strong foundation for intelligent code analysis and can be extended with advanced AI models and multi-language support in future work.

8. TESTING AND VALIDATION

The developed system was tested using multiple sample repositories of varying sizes to evaluate its functionality and reliability. Functional testing was performed on each module, including code parsing, element extraction, summarization, knowledge graph construction, and the Q&A interface. The system successfully processed repositories and generated structured outputs such as extracted functions, classes, and dependency relationships without errors. Validation of the parsing and analysis modules was carried out by comparing the extracted results with the actual source code. The system demonstrated high accuracy in identifying code elements such as functions, classes, and imports. The knowledge graph was also verified to ensure that nodes and edges correctly represented relationships within the repository. The NLP-based summarization module was evaluated qualitatively by comparing generated summaries with the original documentation. The summaries were found to be coherent and informative, although their quality depended on the availability of meaningful input text. Similarly, the Q&A module was tested using different user queries, and it produced relevant responses for keyword-based questions. Overall, the system showed reliable performance for small to medium-sized repositories.

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

While the results were satisfactory, further improvements can be made by incorporating advanced semantic analysis and expanding support for multiple programming languages to enhance accuracy and scalability.

9. CONCLUSION

This paper presented Code Repository Knowledge Extractor, an intelligent system designed to automate the understanding of software repositories. By integrating static code analysis, natural language processing, and graph-based modeling, the system transforms raw source code into structured knowledge and meaningful insights. It successfully extracts key elements such as functions, classes, and dependencies, generates project summaries, and visualizes relationships through a knowledge graph. The system demonstrates that combining AST-based parsing with NLP techniques can significantly improve code comprehension and reduce the effort required for manual exploration. Features such as the Q&A interface and automated onboarding guide further enhance usability by enabling interactive access to repository information and assisting new developers in navigating complex codebases. The results indicate that the proposed approach is effective for small to medium-sized repositories, providing accurate structural analysis and useful summaries. However, the system has certain limitations, including dependence on static analysis and limited support for advanced semantic understanding and multiple programming languages. Overall, the proposed system offers a scalable and extensible solution for automated software repository analysis. It lays a strong foundation for future advancements in intelligent developer tools and highlights the potential of integrating artificial intelligence with software engineering practices to improve productivity and code understanding.

10. FUTURE SCOPE

The proposed system provides a strong foundation for automated code repository understanding; however, several enhancements can be explored to improve its capabilities. One important direction is the integration of deep semantic analysis, enabling the system to understand runtime behavior, control flow, and data flow within programs. This would significantly improve the accuracy of insights generated from the codebase. Another area for future work is multi-language support, allowing the system to analyze repositories written in diverse programming languages beyond Python. Incorporating advanced transformer-based models and large language models can further enhance the quality of summarization and enable more accurate and context-aware question answering. The system can also be extended by integrating real-time GitHub synchronization, enabling continuous analysis of evolving repositories. Additionally, incorporating graph neural networks (GNNs) could improve knowledge graph representation and enable predictive insights such as bug detection or code recommendation. Finally, the development of a more advanced and interactive user interface, along with scalability improvements for handling large-scale repositories, would make the system more practical for real-world applications. These enhancements can transform the proposed system into a comprehensive intelligent assistant for developers and software teams.

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 534-544

References

[1] A. E. Hassan, “The road ahead for mining software repositories,” in Proc. Future of Software Engineering (FOSE), 2007, pp. 48–57.

[2] I. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” in Proc. Int. Conf. Software Maintenance (ICSM), 1998, pp. 368–377.

[3] M.-A. Storey, K. Wong, and H. Müller, “How do program understanding tools affect how programmers understand programs?” in Proc. Working Conf. Reverse Engineering (WCRE), 1997, pp. 12–21.

[4] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, “Automatic generation of natural language summaries for Java classes,” in Proc. Int. Conf. Program Comprehension (ICPC), 2013, pp. 23–32.

[5] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A survey of machine learning for big code and naturalness,” ACM Comput. Surveys, vol. 51, no. 4, pp. 1–37, 2018.

[6] X. Xu, H. Hu, S. Guo, and Y. Chen, “Answering natural language questions over code repositories,” in Proc. ACL, 2020, pp. 1–10.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in Proc. ICLR, 2013.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[9] A. Vaswani et al., “Attention is all you need,” in Proc. NeurIPS, 2017, pp. 5998–6008.

[10] T. Wolf et al., “Transformers: State-of-the-art natural language processing,” in Proc. EMNLP, 2020, pp. 38–45.

[11] F. Chollet, “Deep learning with Python,” Manning Publications, 2017.

[12] NetworkX Developers, “NetworkX documentation,” [13] Explosion AI, “spaCy: Industrial-strength natural language processing in Python,”.

[14] FastAPI, “FastAPI framework documentation,”.

[15] React, “React – A JavaScript library for building user interfaces,” .

[16] J. Leskovec, A. Rajaraman, and J. Ullman, “Mining of massive datasets,” Cambridge University Press, 2014.