International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(Review Article)

Harnessing generative AI to create and understand architecture diagrams

Nishchai Jayanna Manjula * and Akhilesh Dube

Amazon Web Services, USA.

International Journal of Science and Research Archive, 2024, 13(02), 3330-3336

Publication history: Received on 18 November 2024; revised on 24 December 2024; accepted on 26 December 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.13.2.2601

Abstract

Architecture diagrams are required tools for software development, system design, and communication. They facilitate the understanding of complex systems by providing a visual representation of components, relationships, and data flow. However, creating and interpreting these diagrams can be time-consuming and require significant expertise. Generative Artificial Intelligence (AI) offers a potential solution to automate the creation process and improve comprehension. This paper explores how generative AI can be leveraged to automatically generate diverse architectural diagrams from textual descriptions and code repositories. Additionally, the research investigates how AI techniques can assist in understanding and analyzing existing diagrams, thereby easing maintenance, documentation, and stakeholder communication. This paper discusses existing approaches, emerging techniques, challenges, and future directions in this evolving field. Our findings indicate that generative AI can significantly reduce effort in diagram creation and improve analysis while also exploring the limitations of current models.

Keywords: Generative AI; Architecture Diagrams; Software Visualization; AI in Software Engineering; Natural Language Processing; Machine Learning; Diagram Understanding

1. Introduction

Software and system architecture diagrams are fundamental for communication and understanding in various technical domains [1]. These diagrams offer a visual abstraction of a system, illustrating the key components, their interactions, and the overall structure. From high-level conceptual overviews to detailed design plans, architecture diagrams come in diverse forms, catering to a wide range of purposes. They are critical for software design, development, maintenance, documentation, and stakeholder communication [2]. A comprehensive architecture diagram helps software development teams visualize the structure of their project, understand data flows, plan for future expansions and modifications, and discuss design decisions with various audiences. However, the traditional process of manually creating these diagrams is often time-consuming, error-prone, and requires significant domain knowledge. The interpretation of these diagrams is also not always straightforward, particularly for newcomers or for very intricate systems. The lack of accurate or updated documentation that is visually coherent creates challenges, causing ambiguity and delays in development processes. This poses a significant challenge for development teams try to understand complex software structures that evolved during multiple years by several people.

The advent of generative AI presents an opportunity to revolutionize how architecture diagrams are created and understood. Generative models, particularly those leveraging deep learning, can learn complex patterns from data and then generate new examples that resemble those patterns. This technology holds significant potential to automate the process of diagram generation, thereby reducing effort and enhancing the consistency and accuracy of visual representations. Moreover, AI-based techniques can also be used to analyze and interpret existing diagrams, thus assisting teams in gaining deeper insights and identifying potential issues more effectively. Generative AI can potentially transform these tasks of creating, modifying and updating the diagrams to be far more agile and seamless and help in

* Corresponding author: Nishchai Jayanna Manjula

creating self-documenting systems. This includes not just generating a new diagram, but also converting and changing existing diagram formats in various platforms that teams usually use.

This paper aims to explore the application of generative AI for architecture diagram creation and comprehension. It provides an overview of current approaches, challenges, and opportunities and delves into the underlying methodologies, encompassing Natural Language Processing (NLP), computer vision, and various machine learning techniques. We will look at state-of-the-art applications and discuss limitations as well as future directions of this field. This research should contribute to the advancement of this emerging field and to help in understanding and addressing the critical gaps. It will contribute towards realizing the potential of generative AI for increased productivity in software engineering and reducing errors arising from manual documentation of diagrams and maintenance.

1.1. Need of the study

Architectural diagrams provide a visual representation of complex systems, aiding stakeholders in understanding the system's structure and relationships between its different parts [3]. These diagrams are designed for various abstraction levels and purposes, covering various areas of software and system design. Common types of architecture diagrams are explained in detail below:

1.2. Types of Architectural Diagrams

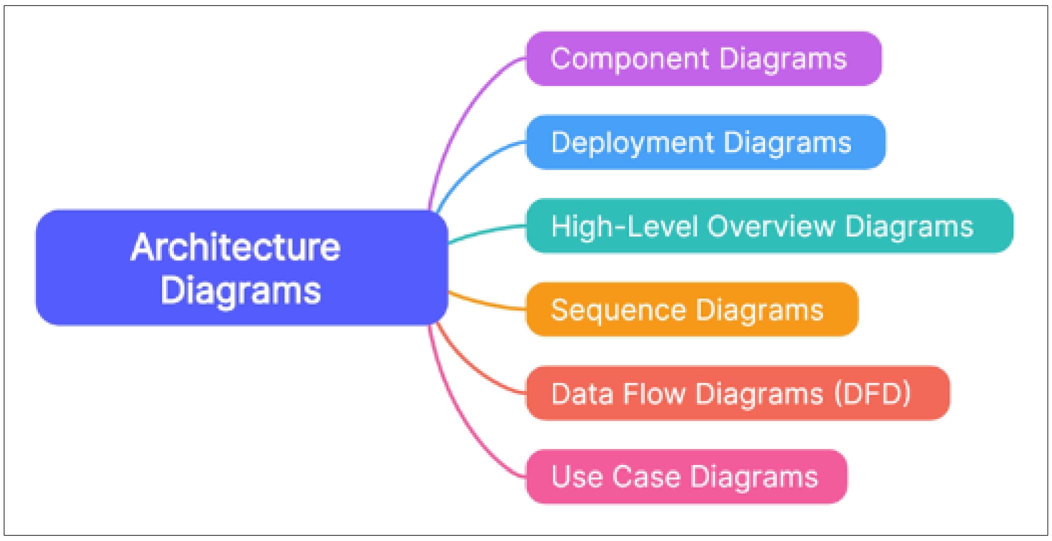

Figure 1 Architectural Diagrams

1.2.1. Component Diagrams

These diagrams illustrate the system's structural makeup by representing its components (modules, classes, libraries) and how they interact through interfaces and dependencies. This type of diagram are crucial for understanding the basic building blocks and is useful in developing a high-level abstraction of how code is structured into various modular chunks.

Component diagrams effectively visualize the division of a system into manageable parts, emphasizing modularity and separation of concerns, which is critical for large scale system design. The representation typically shows different modules as boxes or containers and the interdependencies using arrow connecting them.

1.2.2. Deployment Diagrams

Deployment diagrams represent physical system architecture, such as the placement of software components on hardware (servers, devices) and the network connections between these locations. These diagrams are particularly important when you have different deployments across servers, cloud, mobile or any edge devices.

These diagrams aid in visualizing physical dependencies and connectivity of software and hardware components. They include information about the infrastructure needed for software, the network connections, and locations of deployment, aiding in understanding the hardware environment that supports the application. Deployment diagrams are helpful in system configuration and debugging as well.

1.2.3. Data Flow Diagrams

Data flow diagrams map how data moves across various processes in a system. The source, destination, transformation and the path of data across systems and sub systems are detailed through these diagrams, helping in the understanding of flow and dependencies. These diagrams use distinct shapes to represent data storage, processes and data flow.

Data flow diagrams are valuable for understanding data transformations, user flows, and the flow of information through different system layers, providing valuable insights for developers and business analysts.

1.2.4. UML Diagrams

Unified Modeling Language (UML) provides multiple kinds of diagrams like class diagrams that focus on static structure of code, sequence diagrams that focus on behavior and timing of calls, and many other specific formats. UML is a universal modeling language and its diagrams form the building blocks for large complex systems. UML diagrams, including class diagrams, use case diagrams, and sequence diagrams, offer comprehensive and standardized way to model systems, providing a clear picture of class structures, functionality, and message flow between objects which is essential for creating robust and reliable software designs.

1.2.5. Custom and Informal Diagrams:

In addition to standard types, teams also create custom or informal diagrams to represent system architecture which are helpful in their internal documentation. The lack of consistency and standardized notations in these types can make it difficult for everyone to understand these diagrams unless they have a deep understanding of the context.

These informal diagrams are a big problem in consistency and maintenance for multiple teams and projects. These type of visual representations though quick for local documentation, do not always lend themselves to automated analysis due to lack of clear guidelines.

1.3. **Challenges in Creating and Understanding Architectural Diagrams**

Despite their value, several challenges are associated with architectural diagrams:

1.3.1. Manual Effort

Creating these diagrams manually is often time-consuming, particularly for complex systems. Diagram generation is tedious, requires specialized software and skills to develop an appropriate visual representation. Manual diagram updates also take considerable effort, requiring the engineer or architect to redraw or manually edit. The amount of manual labor needed is multiplied if multiple different kinds of diagrams are needed for a complex architecture. This manual effort often makes it very expensive in terms of manpower and time.

1.3.2. Inconsistency

Without strict guidelines and tools, diagrams can become inconsistent and diverge from the actual implementation, leading to confusion and errors. Over time, systems evolve and unless all changes are updated correctly it is very easy for diagrams to become inconsistent and diverge, thereby becoming a hindrance. The diagrams that do not sync with the actual architecture leads to ambiguity and cause problems to new developers who depend on documentation, and also in communication between development teams.

1.3.3. Interpretation Complexity

Intricate and large diagrams can be challenging to understand, especially for those not deeply familiar with the system architecture or stakeholders who are non-technical. Complex notations, multiple layers and the information that needs to be interpreted creates bottlenecks, thus defeating the fundamental goal of a visual diagram for communication. Complex notations or styles makes diagrams less accessible to non-technical staff, thereby preventing seamless communication with stakeholders.

1.3.4. Maintenance

Updating diagrams to reflect code changes is a maintenance burden that is often overlooked which can result in obsolete diagrams. Keeping the architecture diagrams current with continuous integration and continuous deployment of new changes in code is important but often it is a very difficult and time-consuming process. Manual updating is often skipped, leading to documentation which is out of date and not synced with the system's architecture.

1.3.5. Lack of Accessibility

Informal notations or lack of proper standardized methods creates major accessibility issues. If the diagrams do not follow clear and easily understandable formats, the benefit of visualizations is lost because users and stakeholders struggle to make sense of visual representations of architectures which affects inter-team and inter-stakeholder communication.

1.3.6. Scalability

As systems get complex and large with hundreds or thousands of micro-services or complex sub-systems, the size of architecture diagrams become huge and difficult to scale and manage effectively and efficiently. Very large scale software projects often encounter this problem of difficulty in generating architecture visualizations which represent hundreds or even thousands of micro-services.

These challenges highlight the need for automated and intelligent methods to improve the creation and analysis of architecture diagrams which directly affect software and project management. Generative AI offers potential solutions to mitigate these challenges, leading to a more efficient and productive software development cycle.

2. Generative AI for Diagram Generation

Generative AI models, using techniques such as deep learning, have shown significant potential in producing images, text, and other data types that were previously thought impossible. These techniques, including Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and transformer based models have transformed several industries [4]. The field of architectural diagrams can also benefit from the capabilities of these models in producing more consistent, accurate, and up-to-date visual representations. This section focuses on various methods that harness generative AI to address the aforementioned challenges.

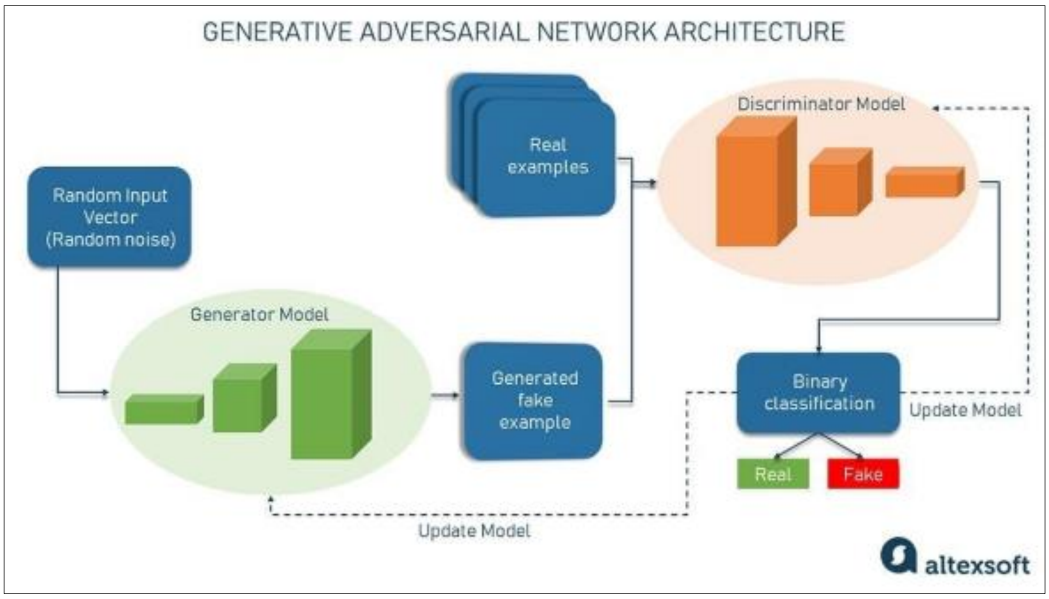

Figure 2 Generative adversarial networks

2.1. AI-Powered Approaches

2.1.1. Text-to-Diagram Generation

This approach uses NLP techniques to analyze textual descriptions of system architectures and transforms them into diagram representations. For example, if an engineering team describes a component, interfaces and relationships through a textual format in documentation, the AI algorithm understands and uses that information to draw a suitable component diagram. Models can be trained on a dataset containing pairs of textual descriptions and their corresponding diagrams to learn to map natural language to visual representations. This method could significantly reduce manual work when creating new diagrams.

- **Transformer-Based Models:** Models like GPT and BERT are adapted for understanding the intricacies of system descriptions and generate appropriate code to build diagrams. The benefit is to not have an intermediary language which requires translating between natural language and machine understandable language. They also retain context for longer textual inputs which makes it possible to describe complex architectures that have multiple relations between components [4]. Transformer models are ideal for this as they can process and interpret textual inputs with complex and varied contexts, helping with building an accurate visual design based on natural language.
- **Semantic Parsing:** With the semantic parsing technique, structured representation of system design can be created from text using semantic role labeling. The structured representation acts as a bridge to build the visual diagrams automatically. This process understands the key nouns and verbs related to a system and uses it to draw the component diagrams or data flow diagrams. Semantic parsing provides precise relationship mapping between textual elements and the graphical structures, ensuring accuracy and coherence in generated diagrams.

2.1.2. Code-to-Diagram Generation

In this approach, AI algorithms directly derive diagrams from source code (e.g. Java, C#, Python) by analyzing software repository (code). It is very useful to generate architecture diagram from existing codebases. The approach requires to be trained on a huge codebase repository which has examples of relationships between code modules, and how data flows and interdependencies [5].

- **Static Code Analysis:** By using static code analysis tools like SonarQube, which look into the code without actually executing them to find the dependencies, modules and data flow relationships, a dependency and system relationship can be drawn. This helps in understanding the flow without actually running the code which enables to keep diagrams in sync with the latest changes and makes debugging of complex dependencies far more easy and manageable. The automation in deriving the architecture from static code improves time efficiency by eliminating manual diagram drawing efforts.
- **Abstract Syntax Tree (AST) Parsing:** Compilers generally use AST for representing the structure and program code. Using AST to build architecture diagram provides highly accurate visualization based on actual program structure. Deep learning techniques help to build these kind of tools by learning and identifying key patterns and dependencies and generating visualizations. Because the AST represents the actual structure of the program as a tree structure, visualization based on AST is very helpful and accurate.

2.1.3. Diagram Synthesis

This area focuses on building diagrams in a specified notation and style given a high-level specification of its structure and components [6]. GANs and VAEs are generally used in this synthesis of diagram approach for training a model that creates an example from scratch or alters an existing one.

- **Generative Adversarial Networks (GANs):** GANs can be trained to generate a diagram by pitting the generator, which tries to create a realistic diagram, and the discriminator, which tries to distinguish between real and generated diagram samples. This method is very effective in generating variations of an existing design and to test different visual representations. Through training GANs to build various diagrams, the AI system can also improve its visual creativity for unique architectural pattern generation.

3. Generative AI Diagram Analysis

Besides the generation of diagrams, generative AI can enhance the interpretation and analysis of existing diagrams through the following methods:

3.1. Automated Diagram Analysis

3.1.1. Image Recognition

Computer vision is used to understand the contents of diagrams and their meaning based on the various shapes used in diagrams. Deep Learning models like CNN, which are initially developed for image classification can be used for pattern matching and object detection from architecture diagrams.

- **Object Detection:** This approach helps to detect various shapes (boxes, arrows, circles) and their textual annotation which makes it possible to extract critical components from a visual representation. Then based on the structure and relations, key components and patterns can be identified. Using object detection on architecture visualizations helps to extract components, their properties and relationships, enabling automation of understanding what an architecture is made of, using a visual as input.

- **Optical Character Recognition (OCR):** OCR tools help to extract text from these diagram images that become useful in generating a comprehensive and structured information from a visual architecture design. These OCR tools generally provide an unstructured representation, and more intelligence needs to be added through NLP techniques to make full use of information contained in diagrams. Extracting textual information through OCR makes the entire content of architecture diagrams searchable and more accessible for further analysis and knowledge extraction.

3.1.2. Diagram Summarization

AI can automate and condense information presented in complex diagrams, highlighting key components and relationships which makes it simpler to interpret huge, detailed diagrams.

- **Graph Neural Networks (GNNs):** Architectural diagrams can be considered as a kind of graph structure with nodes and edges. Graph Neural Network is an ideal choice for learning how components in a system relate to each other from graph based diagrams. Through training models with sufficient datasets of diagrams and relations in them, the AI algorithm can effectively summarize and provide essential context. GNN based summaries are useful in analyzing key patterns in relationships and components.

3.1.3. Diagram Error Detection

Another valuable application of AI is in automated error detection where it flags inconsistencies, logical fallacies, and rule violations.

- **Rule-Based Systems and Constraint Solving**: Rule based system can be incorporated to validate the designs by checking for logical consistencies against established architecture guidelines. Through integration with a constraint solving method, AI tools can quickly find out inconsistencies that could lead to major problem. The rule based system allows customization in a way which aligns with specific organization's practices or industry standards to identify violations of software architecture best practices, through an automated process.
- **Deep Learning Based Anomaly Detection:** Deep Learning model based anomaly detection techniques could help identify deviation from standard design which can help improve consistency. Deep Learning is suitable when we are dealing with different and numerous variations in architectural patterns which cannot be codified through rule-based approach. This makes AI very useful in cases of evolving patterns in architectural designs for a complex system where not every exception and situation can be codified into specific rules.

4. Challenges and Limitations

Despite the significant progress in this field, several challenges remain:

4.1. Data Scarcity

Creating accurate diagrams depends on large annotated data. Finding or building a sufficient dataset of paired text description/code with correct diagram format is difficult and tedious which becomes an important limitation for model accuracy and generality. Data collection and annotation in this specific domain becomes a barrier for adoption.

4.2. Standardization

Lack of clear, comprehensive standards for diagrams limits data compatibility across teams, systems, and organizations, making it harder for AI to perform with consistent effectiveness and output formats. The lack of common standards prevents reuse of pre-trained AI models and algorithms across teams and organizations.

4.3. Context Understanding

AI struggles with understanding contextual information and assumptions made by the designers, resulting in potentially inaccurate or incomplete diagrams. It does not intuitively understand some complex architectural patterns used for specific reasons that cannot be readily understood just from reading textual representation of code or documentation. Architectural diagrams depend heavily on context and intent and these are difficult for AI to understand and represent correctly, often leading to misinterpretations.

4.4. Complex Diagram Generation

Current models are less proficient in handling diverse and complex architectural representations with multi-level layers, multi-modal inputs which creates limitation. Generating visual designs with multiple sub systems and different layers, using varied notations is difficult. Handling the varied type of notations (UML, informal etc.) and dealing with multi-layered design architectures still remains a challenge for Generative AI models.

4.5. Evaluation Metrics

Defining metrics to evaluate the quality of generated diagrams is challenging, making it hard to quantify and improve performance of AI systems and models. There has to be both quantifiable and non-quantifiable ways to evaluate if the generated diagram can serve its primary purpose well. Lack of universally accepted measures for both functionality and aesthetics of diagrams makes it difficult to assess the effectiveness of current AI-driven techniques

5. Conclusion

Generative AI has demonstrated transformative potential in the domain of architecture diagrams by automating their creation and facilitating enhanced understanding and analysis. This automation has the potential to significantly reduce manual effort, improve consistency, aid in maintenance, and enhance communication among teams and stakeholders. By creating a better, faster and more efficient visual and technical communication, AI techniques make development faster.

Current research highlights the use of NLP, computer vision and different AI based models in this field, but also indicates gaps in scalability and handling complex contextual requirements. Future developments will depend on how best those limitations can be mitigated through collaborative and iterative effort by researchers and users in the software and system engineering field. The future direction clearly points towards more sophisticated techniques, closer collaboration, and standardized methods. These efforts are very critical to enable effective leveraging of the full potential of AI tools in producing and understanding architecture diagrams

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Kruchten, P. (2004). The 4+1 view model of architecture. IEEE Software, 21(6), 42-50. https://www.researchgate.net/publication/220018231_The_41_View_Model_of_Architecture.

[2] Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (3rd ed.). Addison-Wesley.

[3] Gorton, I. (2012). Essential software architecture. Springer. https://link.springer.com/book/10.1007/978-3-642-19176-3

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30. https://arxiv.org/abs/1706.03762

[5] Rizk, R., & Lo, D. (2020, October). Automatically generating system architecture diagrams from source code: A deep learning-based approach. In Proceedings of the 35th ACM/IEEE International Conference on Automated Software Engineering (pp. 1023-1027).

[6] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27. https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks