JGTAI
Journal of Global Trends in Artificial Intelligence

Volume 1, No. 1, June 2025-2026
Available online at: www.jgtai.com

Large Language Models in Software Network Analysis: A Survey of Applications, Challenges, and Future Trends

Dr. Amit Jain
Professor
Department of Computer Science and Engineering
OP Jindal University, Raigarh(C.G)
amit.jain@opju.ac.in

Abstract*—Software Network Analysis is a fundamental tool for comprehending the structure, dependencies, complexity and evolution of today's software systems. Traditional analysis techniques are however, not suitable for managing large-scale software repositories and offer limited semantic understanding. Large Language Models (LLMs) have contributed much to software analysis with their newfound abilities in code understanding, reasoning, and automated knowledge generation. This survey provides detailed overview of the integration of LLM with Software Network Analysis including its core principles, recent advancements, key applications, challenges, and research directions. It explores how LLMs can be harnessed for software dependency analysis, code comprehension, vulnerability identification, software architecture analysis, graph reasoning and graph representation learning. It delves further into hybrid models that combine LLMs with GNNs to improve analytical skills and scalability. In addition, the survey identifies several problems: computational complexity, hallucination, explainability, privacy protection, security, and limitations of benchmarks. Lastly, it presents the future trends *like lightweight domain-specific LLMs, multimodal graph-language learning, explainable AI, and autonomous software engineering systems that benefit researchers and practitioners.

Keywords—Large Language Models (LLMs), Software Network Analysis, Software Engineering, Complex Networks, Code Analysis.

I. INTRODUCTION

Network technologies have revolutionized communication, information sharing, and connection with the world. The traditional networks that are heavily dependent on static configuration and manual intervention are the backbone of modern infrastructures but also present major challenges [1]. These networks are difficult to manage and operate, and complex skills are required. However, the difficulty of managing networks is only going to increase as their size and complexity continue to rise [2]. The configuration, optimization, troubleshooting, and security of a network can be compromised due to inefficiencies and human error when the network is large and managed using labor-intensive methods [3]. Software engineering (SE) is regarded as one of the most important endeavors which involves designing, developing, testing, and maintenance of software system in a systematic and predictable way [4]. Nowadays, SE plays a significant role in modern society for the systematic, dependable, and efficient construction of software systems because software is fundamental to numerous industries' infrastructures, including transportation, healthcare, and education.

AI has radically transformed computer technologies over the last decade, moving its way forward in several areas. This progress has spurred the development of AI as a game-changer in the realm of network management, tackling the challenges of conventional approaches [5]. Network configuration, traffic flow optimization, security protocol improvement, and even failure prediction are just a few of the many jobs that may be automated with the use of AI algorithms [6][7]. This drastically cuts down on the opportunities for human mistake and the volume of manual labor required. Particularly important in large-scale, ever-changing contexts, the use of AI in networking improves network security through the automatic detection and response to threats in real-time[8]. A popular strategy for improving AI algorithms is mathematical optimization theory. But AI-powered answers are frequently too complicated and mysterious for regular people to comprehend.

The development of generative AI has been one of the most encouraging advances in the modern era of intelligent machines [9][10]. The remarkable reasoning, generalization, and emergent abilities displayed by text-to-code, text-to-image, and text-to-text LLMs—which have billions of parameters—have generated great interest in both academia and industry [11]. The number of open sources LLMs like GPT-2, LLaMA, and BLOOM is growing rapidly; in fact, ChatGPT alone adds one million users every week. Chip design, protein structure synthesis, and robot embodied intelligence are three areas where domain-adapted LLMs have emerged as useful tools. Generative LLMs may compress information features and vectorize massive amounts of knowledge as tokens, making them capable of assisting or even replacing humans in tasks such as conceptual comprehension, logical reasoning, and decision-making.

A fresh perspective on LLMs in SE is that they can be used to easily reframe numerous SE problems as tasks involving analysis of code, text, repositories, or developer-data. Lots of promising new insights have emerged from applying LLMs to these SE tasks. Tasks like requirements analysis [12]. LLMs have a significant influence on program analysis, code summarization, and the study of textual requirements from various software stakeholders during software design or maintenance. These techniques help overcome the shortcomings of standard static analysis tools, including as imprecision.

A. Structure of the paper

The paper is organised is as follows: Section II provides an overview of software network analysis. Section III discusses fundamentals of Large Language Models. Section IV presents integration of LLMs with software network analysis. The review of relevant literature is covered in Section V. Section VI concludes with a concise synopsis of the findings and recommendations for further research.

II. Software Network Analysis

A graph-based method, Software Network Analysis (SoNA) depicts software systems as interconnected networks. In this model, software entities like classes, modules, packages, or functions are represented as nodes, and their dependencies or interactions are represented as edges [13]. It helps analyze software structure, complexity, maintainability, reusability, and architectural quality. Through software network analysis, graph theory and network metrics, the critical components, failure prone modules, and dependency patterns can be identified, which can be used to optimize software, evolve software, assess software security, and for the intelligent software engineering.

A. Software systems as complex networks

The main areas of intersection between software metrics and complex network theory are: uses of software metrics in software practice, modelling software network growth, measuring software networks, and describing software networks [14]. Detailed below are the sections.

1) Characterization of Software Networks

Software network characterization efforts center on identifying and verifying topological properties (e.g., small world, scale-free, etc.) and investigating software structure-wide commonalities. groundbreaking work in software analysis that first applied complex network theory [15][16]. They dissect the class diagram into an undirected network, with nodes representing classes and edges representing relationships (such as inheritance, association, etc.) between each pair of classes (see Fig. 1).

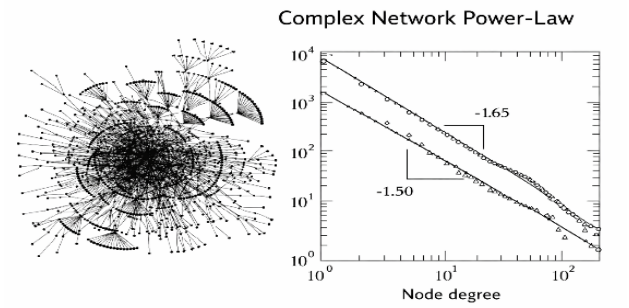

Fig. 1. The software network of JDK 1.2 and its cumulative degree distribution

2) Modeling of Software Networks Growth

A straightforward approach to expanding software systems was put out, predicated on a handful of refactoring methods [17]. The goal of software network evolution modeling is to shed light on the fundamental reasons why refactoring might improve code evolvability through structural changes. A lot of the important parts of the systems that were looked at can be used in this simple refactoring model.

3) Measurement of Software Networks

There are a few ways to measure the structure of software systems that use complicated network theory. In general, these metrics let engineers evaluate specific parts of software architecture and spot faulty software network structures. By defining structural entropy in terms of the degree of nodes, they are able to forecast the software size and construction cost. Additionally, they measure and analyze the software structure's orderliness [18]. Additionally, they laid the groundwork for optimizing software structure by investigating the links between structural entropy, software robustness, and communication efficiency [19].

4) Graph Neural Network

Graph neural networks typically represent non-Euclidean relationships as their processing object. In theory, GNN was founded on the idea of a GNN. Scholars have considered incorporating convolutional operator principles into GNNs, also called graph CCNs [20], following the success of convolutional neural networks. The spectral domain and the spatial domain are the two primary categories of GCNs.

B. Types of software network analysis

A variety of network types are utilized in software network analysis to investigate various relationships within software systems [21]. Software architecture, control flow, data flow, inheritance, and dependency network analyses are some of the most common types. These methods are useful for assessing the completeness, accuracy, complexity, maintainability, and safety of software as well as its overall performance.

1) Software project abstraction

Figure 2 shows how web-based structural module recognition is used to abstract software projects. Enclosing class dependencies that go beyond developer-determined packages is one thing, and a fully-formed module structure that matches the package hierarchy is another [22]. The shown hierarchy not only improves understanding, but it also makes it possible to anticipate the interdependencies among a project's classes [23].

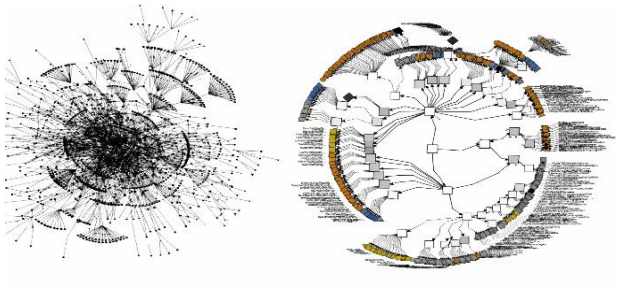

Fig. 2. JUNG framework and Hierarchy of structural modules

2) Software packages refactoring

The refactoring of software packages can also make use of algorithms that detect network modules. Two possible approaches can be used: one is a community discovery strategy, which can reveal extremely modular organization, and the other is a functional module identification technique, which can reveal the underlying functional structure. Common structural module detection algorithms categorize software packages based on the functional and modular relationships between their dependencies (Figure 3) [24].

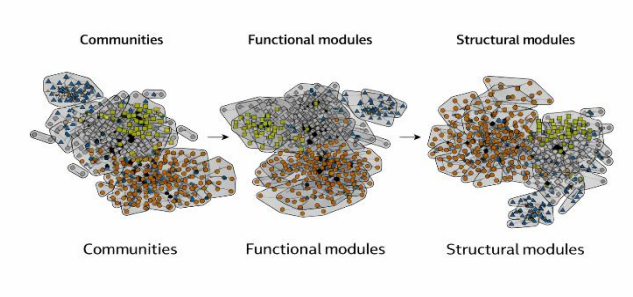

Fig. 3. Communities, Functional and structural modules

3) Software packages prediction

The accuracy of software package predictions for various system classes is displayed in Table I. It should be a node that represents the AI class. After taking into account nodes within the same structural module, the most likely package is then predicted to be the AI package. The technique is used to identify structural modules [25]. For the majority of the systems under consideration, software package predictions have an average probability of over 80%, while for more than 60% of software classes, the full package hierarchy may be resolved with pinpoint accuracy.

TABLE I. NORMALIZED MUTUAL INFORMATION (NMI) BETWEEN SOFTWARE PACKAGES AND IDENTIFIED NETWORK MODULES

Network	MO	CP	MM	GP
flmng	16 / 0.580	14 / 0.609	27 / 0.521	16 / 0.610
colt	19 / 0.519	10 / 0.473	20 / 0.533	19 / 0.530
jung	39 / 0.614	13 / 0.650	30 / 0.661	39 / 0.680
org	47 / 0.503	11 / 0.537	30 / 0.378	39 / 0.536
weka	81 / 0.558	26 / 0.410	49 / 0.430	63 / 0.314
javax	107 / 0.704	59 / 0.761	155 / 0.392	89 / 0.747

4) Software quality indicators

The study's selected quality indicators for software projects and classes are shown in Table II. Use indicators to evaluate project architecture, code reusability and complexity, controllability and vulnerability, information flow, and more. Space constraints prevent us from comparing this method to others that have been developed to evaluate software quality, such as coupling and cohesion metrics.

TABLE II. CLASSIFICATION ACCURACY FOR SOFTWARE PACKAGE PREDICTION

Network	l	l∞	P	P_4	P_3	P_2	P_1
flmng	2.65	4	0.566	←	0.572	0.793	1.000
colt	3.35	4	0.654	←	0.756	0.942	1.000
jung	2.97	4	0.617	←	0.663	0.857	1.000
org	3.50	7	0.616	0.616	0.714	0.989	1.000
weka	3.02	6	0.684	0.692	0.736	0.871	1.000
javax	3.11	5	0.626	0.631	0.816	0.982	1.000

III. FUNDAMENTALS OF LARGE LANGUAGE

LLMs enhance software network analysis by understanding source code, software dependencies, network logs, and architectural relationships using transformer-based learning. They enable automatic code analysis, vulnerability discovery, software documentation and architecture recovery. LLMs offer more intelligent, context-sensitive, scalable analysis than traditional methods, enhancing software quality, security, maintenance, and decision-making in complex software systems.

A. Definition of Large language models (LLMs)

The remarkable powers of LLMs in comprehending, generating, and reasoning in natural language have made them a game-changing technology in artificial intelligence. The crucial domain of networking is just one of several that could be revolutionized by these models, which are trained using massive amounts of textual data [26]. The use of LLMs in computer networks has several potential benefits, including enhanced automation, security, and network design. Both open-source and closed-source LLM models exist. Free and open-source LLMs like Llama, Falcon, and Mixtral let researchers to customize them for different networking tasks.

1) Transformer Architecture

LLMs are primarily designed in the Transformer architecture introduced. This design uses attention and other methods to learn textual relationships over long distances, which allows the model to understand context across entire paragraphs and phrases [27]. The remarkable transformational power of LLMs is attributed to their dependence on self-attention mechanisms that evaluate the importance of individual words in a sequence.

2) Generative Capabilities

The generative nature of LLMs is one of their most powerful features. They are able to produce well-organized, appropriately-structured prose when given only a sentence or two to work with. As a result, LLMs are able to generate stories, answer questions in a natural way, and complete texts [28]. A combination of their training-induced patterns and their natural abilities allows them to mimic human responses. Generative capabilities are utilized in networking to automate tasks such as protocol adaption, configuration generation, and network troubleshooting. LLMs, for example, use past data to offer alternative solutions to network difficulties [29].

3) Pre-training and Fine-tuning

Pre-training and fine-tuning are typically the two steps involved in creating LLMs. Pretraining involves the model learning general language patterns through sentence-level word prediction or context-based word filling [30][31]. A model that comprehends the general structure of language is produced by this step, which entails training on massive data sets. The model can be fine-tuned using smaller datasets learned on during the tuning phase. These datasets can be used to train the model on tasks like translation, summarization, or question-answering.

4) Tokenization

The pre-processing stage of tokenization separates text into non-decomposing parts called tokens. It is essential to LLM training. "Tokens," the linguistic building pieces that differ among models, could be anything from words and subwords to characters. LLMs are able to function because they are able to forecast and create token sequences [32]. The capacity of LLMs to process languages with a wide variety of vocabulary and structures is a key feature. The model's ability to handle unusual or specialized words is greatly affected by the tokenization decision. Tokenization is a crucial process in networking, particularly when dealing with structured data such as configuration scripts, logs, or command-line instructions. These specialized texts can be processed by LLMs with effective tokenization, which either translates them into actionable configurations or identifies errors within them.

5) Few-Shot and Zero-Shot Learning

Long short-term memory (LLM) models are superior to traditional ML models when it comes to few-shot or even zero-shot learning, as opposed to when specific tasks require a large amount of labeled data. This implies they don't need specialized training for new tasks and can instead generalize their skills. An LLM can complete translation or summarization tasks with few examples or no training at all because of the broad knowledge it has acquired during pre-training. In order to train LLMs for networking jobs like translating policy intentions into network settings or producing exact router orders, investigated, prompt engineering methodologies are widely used [33].

6) Scalability

LSTM models perform better when trained on bigger datasets with more parameters. Large models, like GPTs, can generate context-aware and coherent answers since they have hundreds of billions of parameters . This scalability, however, has its drawbacks in terms of energy use and computational expense. The strength of these massive models must be balanced with the necessity for effective deployment on current infrastructure in order to address networking, an area where low latency and real-time performance are paramount.

IV. Integration of LLMs with Software Network Analysis

The integration of LLMs with software network analysis enables intelligent understanding of complex relationships among code components, dependencies, developers, and repositories. LLMs support semantic network representation, code understanding, graph embeddings, and graph reasoning. Combining LLMs with GNNs and hybrid frameworks further enhances software analysis, prediction, vulnerability detection, and automated decision-making.

A. Hybrids of LLMs and other Techniques

Hybridizing, or integrating LLMs with other preexisting software engineering methodologies, showed to be a promising strategy throughout literature review [34]. The literature on code creation with hybrid LLMs is reviewed in this section. A number of writers have created an amalgam of LLMs with planning and search algorithms.

1) Code Understanding.

Code understanding is different from code summarization because it requires a thorough examination of the code in order to comprehend its logic, structure, functionality, dependencies, and the languages, frameworks, and libraries that were utilized. By making use of their robust NL processing capabilities, LLMs may decipher code-related material, including comments and documentation, which can aid in code understanding [35]. The developers may better understand the code's operation, find its dependencies, and provide useful documentation with their help. Developers get greater agency over code optimization, integration, and maintenance with the help of LLMs, which enhance code knowledge through a dual understanding of code and NL.

2) Graph Neural Networks

The GNN is a neural architecture specifically designed to work with data that is already graph-structured, meaning that entities are represented as nodes and relationships are represented as edges. As opposed to sequential models, GNNs propagate information within the graph topology to directly encode relationship dependencies and hierarchies. Structured reasoning tasks such as social network analysis, recommendation systems, biological networks, and program analysis are considerably enhanced by their utilization [36].

3) Large Language Models for Graph Reasoning

The performance of LLMs in graph reasoning has been the subject of numerous efforts to improve LLMs' reasoning capabilities. begins by introducing the NL Graph Benchmark, a tool for assessing how well LLMs perform on different graph reasoning tasks. investigates the effect on LLM performance in graph reasoning tasks of various graph encoding methods and graph structure types [37]. It also introduces GraphQA, which is a new benchmark. Due to the time-consuming nature of writing out graph structures, and can try to encode them using Transformers and then align them with LLMs using GNNs. This method, which takes its cue from the way people process visual information, uses graph structures to create related images, which are then fed into visual LLMs for graph reasoning [38].

4) LLM-based graph representation learning

Several research have used various approaches to create graph embeddings with LM. The two main types of current approaches to graph representation using LLM are supervised and self-supervised methods [39]. Modern supervised methods like Graph Adapter and TAPE are tailor-made for particular graph learning objectives like node classification. OFA and Graph Translator are two examples of additional supervised methods that train models for multiple tasks [40] .

B. Applications of LLMs in Software Network Analysis

1) Vulnerability Detection Problem

This survey's selected research all center on vulnerability detection, the most common application of LLMs in software security. In formal terms, the primary objective is a binary classification issue:

- A vulnerability detector driven by LLM can be represented by and the input source code can be denoted as [41]. A vulnerability status of indicates whether the code is susceptible to attacks or not, as seen in the output.
- vulnerability detection workflow example in the majority of chosen studies, with two subproblems: vulnerability categorization and severity prediction.

2) Software Dependencies

The majority of software systems depend on third-party components that offer functionality through clearly defined interfaces, rather than being built in isolation [42]. Commonly, these connections are called software dependencies. When one piece of software can't do its job properly without the help of another, this is called a software dependency [43].

3) Software architecture Analysis

The components, linkages, interfaces, and interactions that make up a software system are defined by its architecture. Software architecture models offer a transparent framework for system design and development by representing this organization through various structural patterns[44]. These patterns include Layered, Client-Server, Microservices, Service-Oriented, Component-Based, Event-Driven, and Pipe-and-Filter architectures [45].

- An organization's software architecture is its "structure or structures," which include all of the software

components, their attributes that are visible to the outside world, and the connections between them." Software components are typically considered the building blocks when discussing software architecture among practitioners and scholars [46].

C. Challenges in Software Network Analysis Using LLMs

There are several challenges associated with using LLMs for software network analysis, such as the scale and complexity of software networks, the computational burden, data quality considerations, and the ability to accurately represent the relationships between the graph and the language. Reliable and generalized analysis is further hindered by LLM hallucinations, limited explainability, security and privacy concerns, dynamic software evolution, and the absence of standardized benchmarks.

1) Real-Time Processing and Performances

A lot of attention has been paid to the addition of LLMs to network operations and management, but there are still some practical problems to think about. In the first place, issues with processing and performance in real time are apparent. The large compute cycles required by LLMs conflict with the sub-second control loops required by the carrier-grade network. Networks with such a high SLO do not have adequate time for LLM inference delay. For edge devices with limited resources or small network operators, LLMs aren't a viable option due to the increasing computing cost and latency that come along with larger models, even though accuracy generally improves with larger models [47].

2) Hallucination and Output Verification

Hallucination is an inherent challenge that still hampers the application of LLMs for automated intent translation, network analysis, and management. Errors that could result in service outages can occur when certain parameters, such as the QoS level, an invalid interface name, or an incorrect keyword, pass through a closed loop controller. This concern with the hallucination and the semantic conflict is supported with empirical data from the case studies [48]. Similar problems can arise in log-analysis pipelines if the wrong device is targeted due to an incorrect root-cause sentence.

3) Security and Privacy

There are still a lot of privacy and security concerns that prevent LLMs from being used for network management. Network configurations, operational logs, user information, and real-time telemetry are all examples of the sensitive data that can be managed by a network management system. Unfortunately, this makes the LLM a prime target for cyber-attacks or an unwanted source of sensitive information leakage [49][50]. Additionally, LLMs that are trained on domain-specific data sets may inadvertently retain and reveal sensitive information, such as network topology, configurations, or user access policies. Attackers may be able to exploit vulnerabilities in such a way that they expose or reconstruct sensitive information from model outputs, which could be inference attacks.

4) Scalability of Software Systems

A software system's scalability, or its capacity to manage growing loads of work or traffic, is an essential component of software engineering [51]. As the system's workload grows or the number of users grows, the system can function efficiently and effectively [52]. The ability to scale to handle massive amounts of data and traffic is essential for software systems that have numerous users. For software systems, the two most prevalent scaling patterns are horizontal and vertical. Adding additional machines to the system is one way to scale up horizontally so that it can handle greater traffic.

V. Literature Review

Recent research has shown that LLMs can be valuable for software network analysis, automating tasks such as traffic classification, log analysis, vulnerability detection, anomaly detection, and network configuration.

M. Rashid, et al., (2026) provides solutions to technical issues including bias in training data, difficulties in comprehending model outputs, and possible adversarial attacks. Accountability, compliance, and responsible use are also considered from an ethical and regulatory perspective. The survey integrates the results of recent research and industry experience with the outlook for creating more secure models, designing more robust defensive measures, and designing policy that incorporates innovation while maintaining security. The paper concludes that LLMs can improve software security, but only if their use is cautious, with a focus on minimizing the risks associated with them through teamwork, supervision, and continuous improvement [53].

S. J. Jananloo, (2025) LLM, on the other hand, are used in software engineering to do things like writing code, reviewing code, documenting code, and even finding bugs. The possibility of automating STA has piqued the interest of researchers in these capabilities. There is a growing demand to maximize the potential of LLMs in STA since their performance is still insatiable. There are a lot of papers that talk about how to use LLMs for SVD but as far as know, none of them have looked at how well they can do STA [54].

J. Kar, et al., (2025) provides a novel approach to automating the transformation of power system dynamic models between different formats by making use of LLMs. The goal is to automate the tedious and error-prone process of moving models between various software environments by capitalizing on LLMs' textual understanding and processing capabilities. Automating the process of importing model parameters from commercial software like PSS/E into open-source platforms like Dynawo makes it easier for academics and engineers to employ a larger choice of tools for power system analysis and design [55].

A. Hatahet, et al., (2025) suggest integrating LLM with reverse engineering (RE) methods to construct SADs automatically from source code. Using a combination of a thorough component diagram extraction, prompt-guided LLM analysis to weed out architecturally significant aspects, and state machine diagrams generated from code logic to represent component behavior, and method is able to restore both static and behavioral views. This two-view design is a scalable and sustainable solution to the problem of manual architectural documentation [56].

M. K. Aslan, et al., (2025) two distinct transformer-based approaches are utilized: the encoder-only Code BERT and the decoder-only Qwen-2.5Coder. The optimal models are tested

on two benchmark datasets: PrimeVul and Bi Vul, which are very different in data duplication and label quality. The BigVul benchmark shows that Qwen-2.5-Coder beats Code BERT, but on the realistic and deduplicated PrimeVul dataset, both models perform far worse. In particular, Qwen-2.5-Coder is quite sensitive to high-quality samples; it achieves a recall of only 2.37 percent, indicating that models that rely solely on decoders may overfit when faced with noisy or redundant data [57].

D. Song et al., (2024) launch a preliminary investigation and suggest LUNA, a universal LLM analysis framework that is both general and extensible, allowing for the customizable and human-interpretable examination of LLMs from various quality viewpoints. With the help of the many abstract model construction methods included into LUNA, begin by using data from the necessary trustworthiness viewpoints to build an abstract model that can serve as a proxy and auxiliary asset for analysis. The goal of gathering and establishing a variety of assessment metrics is to evaluate the abstract model's quality on two levels: the abstract model level and the semantics level [58].

Z. Li, et al., (2024) TELLMe, a new two-stage method for teaching and exploiting large language models, was proposed to address the MS. In the first step, make effective use of the extensive domain knowledge of LLMs to retrieve models. To address the limitations of LLMs when it comes to MS task understanding, incorporate ICL and CoT. Step two uses TE to rate models even further after step one retrieves possible high-performing models [59].

Research gap: The existing literature demonstrates that LLMs have advanced software network analysis, vulnerability detection, software trustworthiness assessment, architecture recovery, model conversion, and automated code analysis. However, most studies focus on isolated applications rather than unified frameworks integrating security, performance, and explainability. Limited attention has been given to real-time network environments, scalable deployment, cross-platform interoperability, and resource-efficient LLMs. Moreover, issues like hallucination, adversarial robustness, privacy preservation and standardized evaluation benchmarks are yet to be sufficiently addressed. Development of trustworthy, explainable, lightweight LLM-based frameworks for safe, adaptive, and reliable software network analysis in varied operating situations requires future research.

TABLE III. LITEREATURE STUDY OF LARGE LANGUAGE MODELS IN SOFTWARE NETWORK ANALYSIS

Reference	Study On	Approach	Key Findings	Challenges	Future Direction
M. Rashid et al. (2026)	LLMs in Software Security	Survey of LLM applications, security risks, ethical and regulatory aspects	LLMs improve software security but introduce bias, adversarial risks, and governance concerns	Model bias, explainability, adversarial attacks, compliance	Develop secure, explainable LLMs with stronger defense mechanisms and regulatory frameworks
S. Jananloo (2025)	Software Trustworthiness Assessment (STA)	Evaluation of LLMs for code generation, review, documentation, bug detection, and STA	LLMs show potential for STA, but performance is still insufficient	Limited STA accuracy and lack of comprehensive evaluation	Improve LLM-based trustworthiness assessment and optimize vulnerability analysis
J. Kar et al. (2025)	Power System Model Conversion	LLM-based automated conversion between PSS/E and Dynawo formats	Reduces manual effort and improves interoperability between software platforms	Conversion accuracy and compatibility across diverse models	Extend support for additional power system tools and improve automation reliability
A. Hatahet et al. (2025)	Software Architecture Documentation	Reverse engineering with LLM-guided extraction of component and state diagrams	Automates scalable generation of software architecture documentation	Identifying architecturally significant components and maintaining accuracy	Enhance behavioral analysis and support large-scale software systems
M. K. Aslan et al. (2025)	Software Vulnerability Detection	Comparison of CodeBERT and Qwen-2.5Coder using PrimeVul and BigVul datasets	Qwen-2.5Coder performs better on BigVul but degrades on realistic datasets	Overfitting, low recall, sensitivity to data quality	Develop robust LLMs using high-quality datasets and better generalization techniques
D. Song et al. (2024)	LLM Quality Analysis (LUNA)	Universal framework using abstract models and multi-perspective evaluation metrics	Provides interpretable assessment of LLM quality and trustworthiness	Complexity of abstract model construction and semantic evaluation	Expand quality metrics and improve scalability for diverse LLM applications
Z. Li et al. (2024)	Model Selection using LLMs (TELLMe)	Two-stage framework combining LLM prompting, ICL, CoT, and Transferability Estimation	Improves retrieval and ranking of machine learning models	LLM understanding of model selection tasks and transferability limitations	Refine transferability estimation and enhance automated model recommendation systems

VI. CONCLUSION AND FUTURE WORK

LLMs are becoming a revolutionary tool to advance the field of software network analysis by delivering intelligent, context-aware, and scalable solutions to comprehending complex software systems. This survey summarizes the core principles of SoNA and LLMs and their integration for analyzing software, including code understanding, vulnerability detection, software dependency analysis, architectural analysis, and graph-based reasoning. The survey also explored the evolving trend of integrating LLMs with other AI models, like GNNs, to enhance software quality, maintainability, and security through hybrid solutions. Although these advances are encouraging, there are still several hurdles to overcome: computational burden, hallucination, limited explainability, privacy and security issues, scalability concerns, and the absence of agreed upon evaluation benchmarks. It is crucial to overcome these limitations to facilitate the deployment of LLM-based software analysis systems in real-world environments in a reliable and trustworthy manner. Future work should include the development of lightweight and explainable LLMs, multimodal graph-language learning frameworks, efficient

real-time analysis approaches, and benchmark datasets. In sum, the marriage of LLM to software network analysis is a promising direction of research that will likely help to transform the field of intelligent software engineering, automated software maintenance, and next-generation software QA.

REFERENCES

[1] N. Van Viet and N. T. Vinh, "Large language models in software engineering," *J. Educ. Sustain. Innov.*, vol. 2, no. 2, pp. 146–156, 2024, doi: 10.56916/jesi.v2i2.968.

[2] C.-N. Hang, P.-D. Yu, R. Morabito, and C.-W. Tan, "Large Language Models Meet Next-Generation Networking Technologies: A Review," *Futur. Internet*, vol. 16, no. 10, p. 365, Oct. 2024, doi: 10.3390/fi16100365.

[3] N. Adik, "The Rise of Open and Free Networks: A Community-Driven Paradigm for Decentralized Connectivity," in *2025 IEEE First International Conference on Innovations in Engineering and Next-Generation Technologies for Sustainability (ICINVENTS)*, Coimbatore, India: IEEE, 2025, pp. 1–9, November. doi: 10.1109/ICINVENTS64613.2025.11402224.

[4] Q. Zhang *et al.*, "A survey on large language models for software engineering," *Sci. China Inf. Sci.*, vol. 69, no. 4, p. 141102, 2026.

[5] V. K. R. K. Koppireddy, "Passwordless Authentication: A Modern Approach to Secure Access," *J. Inf. Syst. Eng. Manag.*, vol. 10, no. 63, 2025.

[6] R. Tang, Y. N. Chuang, and X. Hu, "The Science of Detecting LLM-Generated Text," *Commun. ACM*, vol. 67, no. 4, pp. 50–59, 2024, doi: 10.1145/3624725.

[7] K. K. Mohammed, "Leadership Practices of Data Engineering for AI and Machine Learning," *Int. J. Sci. Res. Eng. Trends*, vol. 12, no. 1, 2026, doi: 10.5281/zenodo.18677279.

[8] S. R. Somula, "Mitigating Estimation Challenges in Large-Scale Projects: An Integrated Framework for Enhanced Accuracy and Adaptability," *IPHO-Journal Adv. Res. Sci. Eng.*, vol. 3, no. 10, pp. 01–09, 2025, doi: https://doi.org/10.5281/zenodo.17462211.

[9] Y. Huang *et al.*, "Large Language Models for Networking: Applications, Enabling Techniques, and Challenges," *IEEE Netw.*, vol. 39, no. 1, pp. 235–242, Jan. 2025, doi: 10.1109/MNET.2024.3435752.

[10] M. Mohit, "The Rise of Generative AI: Evaluating Large Language Models for Code and Content Generation," *Int. J. Adv. Sci. Eng. Technol.*, vol. 10, no. 4, pp. 20643–20649, 2023.

[11] J. Cao, M. Li, M. Wen, and S. C. Cheung, *A study on prompt design, advantages and limitations of ChatGPT for deep learning program repair*, vol. 32, no. 1. Association for Computing Machinery, 2025. doi: 10.1007/s10515-025-00492-x.

[12] N. Tihanyi, Y. Charalambous, R. Jain, M. A. Ferrag, and L. C. Cordeiro, "A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification," *Proc. - 2025 IEEE/ACM Int. Conf. Autom. Softw. Test, AST 2025*, pp. 136–147, 2025, doi: 10.1109/AST66626.2025.00020.

[13] R. K. Kanneganti, "Security and Compliance Issues in Cloud-Based Deployments of Content and Workflow Management Systems," *Eastasouth J. Inf. Syst. Comput. Sci.*, vol. 2, no. 01, pp. 131–138, Aug. 2024, doi: 10.58812/esiscs.v2i01.1122.

[14] K. Jangiti, "Design and Validation of a Machine Identity Governance Framework for AI Agents in Multi-Cloud Environments," in *SoutheastCon 2026*, IEEE, Feb. 2026, pp. 1–6. doi: 10.1109/SoutheastCon63549.2026.11476363.

[15] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Phys. Rev. E - Stat. Physics, Plasmas, Fluids, Relat. Interdiscip. Top.*, vol. 68, no. 4, 2003, doi: 10.1103/PhysRevE.68.046116.

[16] A. Joon, B. K. R. Janumpally, A. Gogineni, and P. Chatterjee, "Efficient Large-Scale Intrusion Identification and Prevention in Distributed Cloud Networks Using Artificial Intelligence," in *2025 5th International Conference on Intelligent Technologies (CONIT)*, 2025, pp. 1–8. doi: 10.1109/CONIT65521.2025.11167760.

[17] R. Sole and S. Valverde, "Information Theory of Complex Networks: On Evolution and Architectural Constraints," *http://www.santafe.edu/research/publications/workingpapers/03-11-061.pdf*, vol. 650, 2004.

[18] Y.-T. Ma, K.-Q. He, B. Li, J. Liu, and X.-Y. Zhou, "A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems," *J. Comput. Sci. Technol.*, vol. 25, no. 6, pp. 1184–1201, 2010, doi: 10.1007/s11390-010-9398-x.

[19] K. Chatterjee *et al.*, "HoST: integrating Heuristic knowledge with attention-based LSTM networks for Thunderstorm prediction," *Sci. Rep.*, Jul. 2026, doi: 10.1038/s41598-026-57889-1.

[20] N. A. Asif *et al.*, "Graph Neural Network: A Comprehensive Review on Non-Euclidean Space," *IEEE Access*, vol. 9, pp. 60588–60606, 2021, doi: 10.1109/ACCESS.2021.3071274.

[21] S. D. Veeravalli, S. B. Venkata, A. Ashirova, R. MS, S. A. D. Poovaiah, and D. Turaev, "Breaking the Swarm Paradigm: Single Candidate Optimization for Cloud Task Allocation," in *2025 International Conference on Innovations in Intelligent Systems: Advancements in Computing, Communication, and Cybersecurity (ISAC3)*, Bhubaneswar, India: IEEE, 2025, pp. 1–7, July.

[22] S. K. Malaraju and S. K. Madishetty, "AI-Augmented Compiler Optimization for Energyefficient Software Execution on Embedded Systems," in *2025 14th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, Uttar Pradesh, India: IEEE, 2025, pp. 1–6, November. doi: 10.1109/SMART66937.2025.11389313.

[23] L. Šubelj and M. Bajec, "Clustering assortativity, communities and functional modules in real-world networks," 2012.

[24] L. Šubelj and M. Bajec, "Community structure of complex software systems: Analysis and applications," *Phys. A Stat. Mech. its Appl.*, vol. 390, no. 16, pp. 2968–2975, 2011, doi: https://doi.org/10.1016/j.physa.2011.03.036.

[25] L. Šubelj and M. Bajec, "Ubiquitousness of link-density and link-pattern communities in real-world networks," *Eur. Phys. J. B*, vol. 85, no. 1, p. 32, 2012, doi: 10.1140/epjb/e2011-20448-7.

[26] T. B. Brown *et al.*, "Language models are few-shot learners," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, 2020, doi: 10.65525/svup.9788199778009.2026.224-230.

[27] A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, 2017.

[28] S. Hoppe and M. Toussaint, "Qgraph-bounded Q-learning: Stabilizing Model-Free Off-Policy Deep Reinforcement Learning," 2020.

[29] C. Shekhar Pareek, "Chaos Testing: A Proactive Framework for System Resilience in Distributed Architectures," *Int. J. Sci. Res.*, vol. 13, no. 11, pp. 851–855, Nov. 2024, doi: 10.21275/SR241110081650.

[30] N. Ding *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nat. Mach. Intell.*, vol. 5, pp. 1–16, 2023, doi: 10.1038/s42256-023-00626-4.

[31] S. R. Chanthati, "Integrating Large Language Models (LLMs) in Cognitive Medical Robotics," in *International Conference on Artificial Intelligence and Robotics (AIR-2025)*, Zenodo, 2025, p. May. doi: 10.5281/zenodo.20316245.

[32] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Comput. Surv.*, vol. 55, no. 9, Jan. 2023, doi: 10.1145/3560815.

[33] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," *Adv. Neural Inf. Process. Syst.*, vol. 35, no. NeurIPS, 2022.

[34] A. Fan *et al.*, "Large Language Models for Software Engineering: Survey and Open Problems," *Proc. - 2023 IEEE/ACM Int. Conf. Softw. Eng. Futur. Softw. Eng. ICSE-FoSE 2023*, pp. 31–53, 2023, doi: 10.1109/ICSE-FoSE59343.2023.00008.

[35] D. Shen, X. Chen, C. Wang, K. Sen, and D. Song, "Benchmarking Language Models for Code Syntax Understanding," *Find. Assoc. Comput. Linguist. EMNLP 2022*, pp. 3071–3093, 2022, doi: 10.18653/v1/2022.findings-emnlp.243.

[36] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017.

[37] V. Chaturvedi, "Modern Software Development with Java , Spring Boot , and Python : A Survey of Frameworks and Best Practices," *ESP J. Eng. Technol. Adv.*, vol. 3, no. 4, pp. 188–197, 2023, doi:

10.56472/25832646/JETA-V3I8P121.

[38] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Can Language Models Solve Graph Problems in Natural Language?," *Adv. Neural Inf. Process. Syst.*, vol. 36, no. NeurIPS, pp. 1–22, 2023, doi: 10.52202/075280-1345.

[39] Y. Muni, "Advanced Multi-Protocol Label Switching (MPLS)-Enabled Networks: A Survey of Emerging Approaches Via Traffic Engineering," *Eng. Technol. J.*, vol. 11, no. 01, Jan. 2026, doi: 10.47191/etj/v11i01.22.

[40] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi, "Harnessing Explanations: Llm-To-Lm Interpreter for Enhanced Text-Attributed Graph Representation Learning," *12th Int. Conf. Learn. Represent. ICLR 2024*, pp. 1–22, 2024.

[41] S. Singh, "Advancing Wireless Communications with Open Radio Access Network," in *2025 6th International Conference on Data Intelligence and Cognitive Informatics (ICDICI)*, IEEE, Jul. 2025, pp. 682–687. doi: 10.1109/ICDICI66477.2025.11135206.

[42] H. P. Cyril, N. D. Bhandarwar, S. Kumara, and S. Mathur, "Securing Containerised Applications Using Kubernetes-Based Orchestration in Software Development," in *2026 International Conference on Intelligent Computing, Networks, and Security (IC-ICNS)*, 2026, pp. 1–6. doi: 10.1109/IC-ICNS68863.2026.11537887.

[43] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empir. Softw. Engg.*, vol. 24, no. 1, pp. 381–416, Feb. 2019, doi: 10.1007/s10664-017-9589-y.

[44] M. Mittal, "The Metaverse and Web3: A New Era of Decentralized Digital Ecosystems," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 10, no. 06, Jun. 2022, doi: 10.15680/IJIRCCE.2022.1006001.

[45] R. Dandigam, R. T. Thutari, and T. Vaidya, "LLM-Augmented Agentic Consensus Swarms for Autonomous Edge Security," in *2026 14th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, Mar. 2026, pp. 1–8. doi: 10.1109/ISDFS69419.2026.11459110.

[46] S. Becker, "Coupled model transformations for QoS enabled component-based software design," p. 3, 2008.

[47] G. Yang, Q. Ye, and J. Xia, "Unbox the black-box for the medical explainable AI via multi-modal and multi-centre data fusion: A mini-review, two showcases and beyond," *Inf. Fusion*, vol. 77, pp. 29–52, 2022, doi: https://doi.org/10.1016/j.inffus.2021.07.016.

[48] X. Lian *et al.*, "Configuration Validation with Large Language Models," no. 1, pp. 1–12, 2023.

[49] L. Huang *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Trans. Inf. Syst.*, vol. 43, no. 2, 2025, doi: 10.1145/3703155.

[50] R. Taware, S. K. Anumula, H. R. Chennam, and O. H. Kundurthy, "Exploring AI-Based Self-Configuring Slice Allocation Across Hybrid Access Networks," in *2025 Third International Conference on Networks, Multimedia and Information Technology (NMITCON)*, IEEE, Aug. 2025, pp. 1–6. doi: 10.1109/NMITCON65824.2025.11188257.

[51] J. B. Mehta, "Ai-Driven Test Engineering For Cloud-Native Systems," *Int. J. Data Sci. IoT Manag. Syst.*, vol. 5, no. 1, Jan. 2026, doi: 10.64751/ijdim.2026.v5i1.297.

[52] K. Shivashankar, G. S. Al Hajj, and A. Martini, "Scalability and Maintainability Challenges and Solutions in Machine Learning: Systematic Literature Review," pp. 1–45, 2025.

[53] M. B. Rashid *et al.*, "A Survey on Large Language Models in Software Security: Opportunities and Threats," *Computers*, vol. 15, no. 4, pp. 1–26, 2026, doi: 10.3390/computers15040226.

[54] S. J. Jananloo, "Software Trustworthiness Assessment via Large Language Models (LLMs)," in *2025 20th European Dependable Computing Conference Companion Proceedings (EDCC-C)*, 2025, pp. 72–75. doi: 10.1109/EDCC-C66476.2025.00034.

[55] J. Kar, J. Pan, and J. Poland, "Power System Model Conversion using Large Language Models," in *2025 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, 2025, pp. 1–5. doi: 10.1109/ISGTEurope64741.2025.11305583.

[56] A. Hatahet, C. Knieke, and A. Rausch, "Generating Software Architecture Description from Source Code using Reverse Engineering and Large Language Model," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2025, pp. 566–575. doi: 10.1109/MODELS-C68889.2025.00080.

[57] M. K. Aslan, Y. E. Alkan, M. B. Alican, and Ö. Özdemir, "Utilizing Large Programming Language Models on Software Vulnerability Detection," in *2025 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2025, pp. 1–6. doi: 10.1109/ASYU67174.2025.11208282.

[58] D. Song *et al.*, "LUNA: A Model-Based Universal Analysis Framework for Large Language Models," *IEEE Trans. Softw. Eng.*, vol. 50, no. 7, pp. 1921–1948, 2024, doi: 10.1109/TSE.2024.3411928.

[59] Z. Li, J. Bai, Z. Chen, C. Li, Y. Ouyang, and W. Rong, "TELLMe: Teaching and Exploiting Large Language Models for Model Selection in Text Retrieval," in *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024, pp. 1–8. doi: 10.1109/IJCNN60899.2024.10651417.