IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 12, Issue 9, September 2023

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.423

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

Exploring the Role of Large Language Models in Automated Code Review and Software Quality Enhancement

Gopinath Kathiresan

Apple Inc, USA

ABSTRACT: As the newest, most savvy technology domain, Large Language Models (LLMs) are the technological force which is now revolutionizing Automated Code Review and Software Quality Assurance, making the context aware analysis, adaptive learning, and the intelligent recommendations. However, traditional code review methods are not only time consuming, but also prone to human error and are not scalable. The use of LLMs in software engineering allows for the detection of defects, optimization of performance and the maintenance of improve mental quality. In this study, we examine the path of automated code review, LLM capabilities and the effect they have on software quality. LLMs clearly have great advantages, but accuracy is still a concern, false positives abound, biases lurk, and computation can be a constraint. The research also identifies areas of future advancement on the hybrid AI-human review systems as well as improved LLM architectures that understand the code significantly better for more robust reviews.

KEYWORDS: Large Language Models, Automated Code Review, Software Quality, AI in Software Engineering, Predictive Analytics

I. INTRODUCTION

1.1 Background on Software Quality and the Importance of Code Review

Software quality is an important part of software development, being responsible for the high reliability, maintainability, as well as security for a system. High quality software will decrease defects, increase performance and enhance user satisfaction. The problem of software quality becomes increasingly difficult in modern software systems with growing complexity. Code review is most effective when used as one of the ways to keep the software quality high. Their crucial contribution in detecting defects early time, lowering technical debt and to follow coding standards (McIntosh, et al. 2016).

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

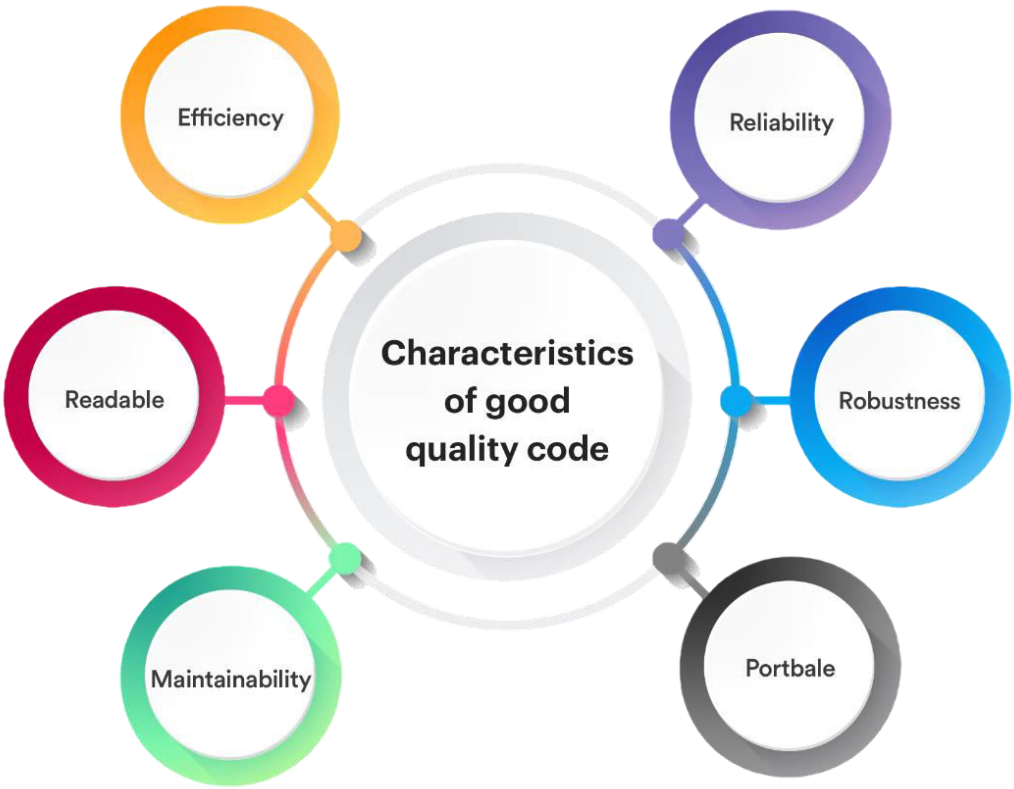

Figure 1: Characteristics of good quality code

Source: Hiren Dhaduk (2019) Code quality: Its Importance in Custom Software Development. https://www.simform.com/blog/code-quality/

Traditionally code reviews were done manually by experienced developers who really looked at every line of code. This method works well, but it is also both time consuming and requires many hours of work from developers who have to devote man hours to the code review and other code creation. Additionally, manual reviews are susceptible to human error and inconsistency as different reviewers may possess differing levels of expertise and precision in their reviews. With these challenges, the industry starts to use the automated code review tools in order to find common mistakes and enforce best practices. While many current automated tools use rule-based approaches, they are less flexible and intelligent compared to those required to deal with the complex patterns of programming and ever-changing best practices (Fan et al., 2023). There is this gap, which has facilitated the integration of Large Language Models (LLMs) in software engineering, in particular, automated code review and software quality improvement.

1.2 Introduction to Large Language Models (LLMs) and Their Role in Software Engineering

Large Language Models (LLMs) attract major interest from researchers because they demonstrate abilities to emulate human text processing along with text creation. Large Language Models which receive their training from extensive code and natural language databases enable superior pattern detection and logical processing functions for various programming-related operations. The ability of LLMs to adapt to different coding methods along with new programming methods differs from conventional static analysis tools (Chen et al., 2021).

Beyond code review roles LLMs perform various operations in the field of software engineering. The models find applications in multiple software engineering tasks including code completion, bug detection, security vulnerability assessment, code summarization and automated programming exercise production (Sarsa et al., 2022). The capacity of LLMs to process extensive code along with their capability to discover faint patterns stands out as their main benefit. The analysis of code functionality by LLMs reveals potential operational performance problems together with security weak points and practice deviation. The feedback recommendations made by LLMs include natural language explanations which assist developers to gain insight from the suggestions through explanations (Austin et al., 2021).

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

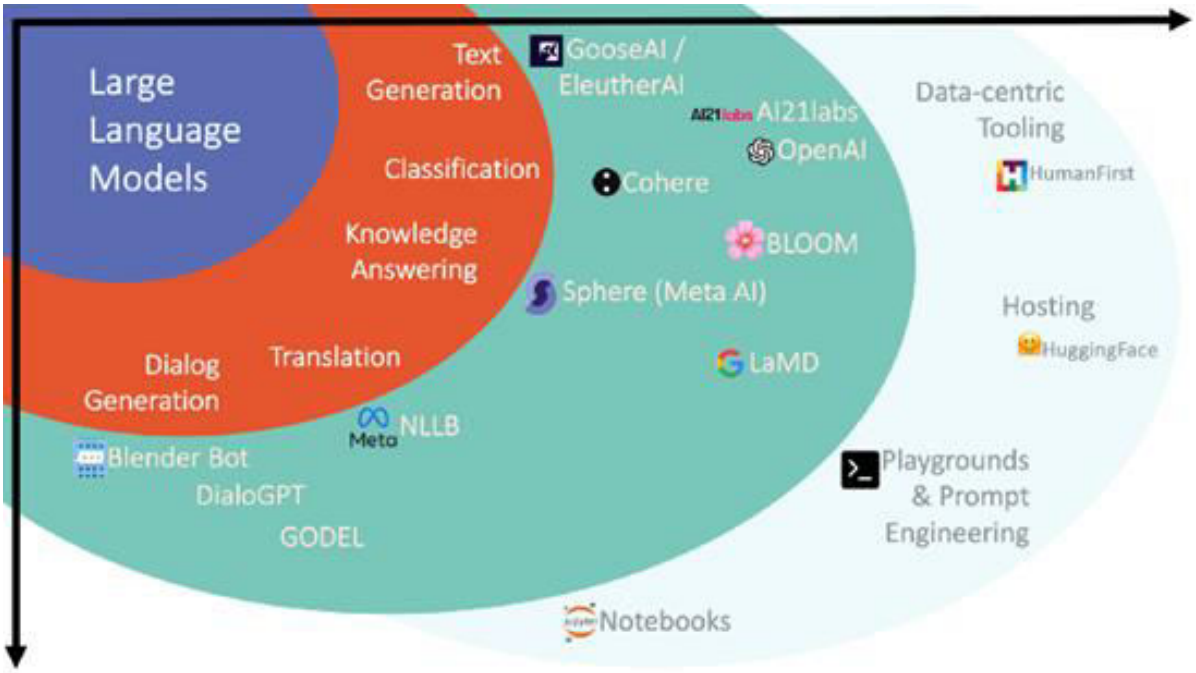

Figure 2: The present state of the **Large Language Model (LLM)** landscape in terms of features, products, and supporting software.

Source: Shenwai D. S., (July 4, 2023). What are Large Language Models (LLMs)? Applications and Types of LLMs
The automated code review process benefits from LLMs to both analyze contexts dynamically and develop through adapted training methods. The continuous improvement of LLMs occurs through updated dataset training which enables these systems to maintain their relevance in changing development settings against manual updates needed for rule-based systems. The models create smart code comments and perform code optimization while recommending different execution methods to improve code performance together with maintainability (Wang et al., 2023). Software development teams that use AI-driven tools will experience a complete transformation in their quality assurance process through LLM application in code reviews.

1.3 Objective and Significance of the Study

The research aims to evaluate Large Language Models' contribution to automated code review and software enhancement specifically regarding defect discovery and code optimization and developer effectiveness. Research examines current developments in Large Language Models alongside their software engineering applications to explore AI tools that boost software quality and automation of development processes and human feedback decision support.
The significance of this study is in the fact that AI is becoming increasingly important in software engineering. Given that organizations are continually seeking to build scalable and secure systems, being able to automate and improve code review processes will be a differentiator. An application of LLMs will reduce cognitive load of developers, reduce human error and accelerate the software development cycle. In addition, this study analyzes real world case studies and empirical research into such LLM based code review systems with aim of deriving actionable best practices in terms of integrating AI driven tools into modern software development environments (Zhou et al., 2022).

II. THE EVOLUTION OF AUTOMATED CODE REVIEW

2.1 Traditional Code Review Processes and Challenges

Code review has always been a part of software development culture to enhance code quality, keep everything consistent, and also reduce defects before deployment. Traditionally, the process has been manual, in which developers do peer reviews, in which they look for logical errors in each other's code, for security vulnerabilities, and for coding standards. Manual reviews are quite effective, but there are numerous problems to be had with time constraints, human bias and scalability issues (McIntosh et al., 2016).

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

Among the main challenges of manual code reviewing is belonging to the experienced developers. Such an inconsistency can be a problem for junior developers to detect subtle issues resulting in mediocre review quality across teams. Furthermore, the process can take time and delays software releases especially in the agile environments where frequent update is needed (McIntosh et al., 2014). The second issue is review fatigue which is when repeated exposure to similar patterns result in reviewers no longer to detect important errors, in effect raising the chance that defects will make their way into production.

2.2 The Rise of Automation in Software Quality Assurance

Automated code review tools have also been implemented to overcome this challenge and increases the efficiency and accuracy. The early automation efforts involved making use of static analysis tools like SonarQube and ESLint based on existing code for syntax error, security vulnerabilities and also style adherence. These tools enforce coding best practices but only have a small degree of capacity in both understanding the context and intent of the code (Fan et al., 2023).

Unit testing frameworks were enhanced with automated unit testing frameworks that let developers validate code functionality automatically. Nevertheless, while these tools catch syntactic and logical errors, they do not possess deeper contextual understanding, i.e., detect how subtle security flaws or design inefficiencies (Wang et al., 2023).

2.3 Early Approaches to Automated Code Review and Their Limitations

Initial automated code review systems were based on the rule based engines, where the errors were detected via manually defined heuristics. These systems worked well catching the simple errors like missing semicolons or unused variables, but not so well with complicated logical flaws. Moreover, we also observed that these tools could easily result in unwanted false positives when identifying problem code (Sarsa et al., 2022).

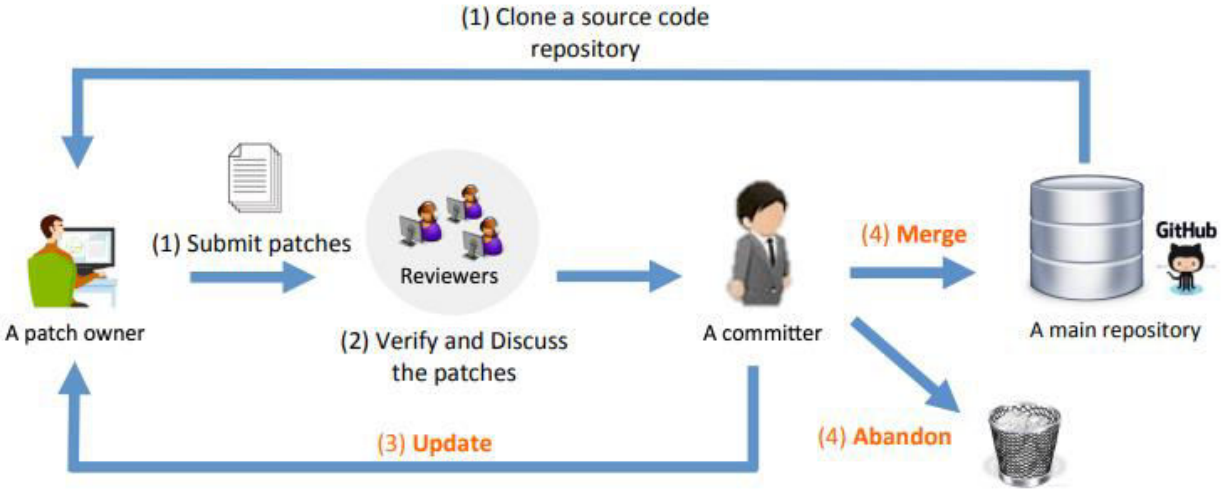

Figure 3: An Overview of Modern Code Review Processes

Source: Toshiki Hirao, Akinori Ihara, Yuki Ueda, Passakorn Phannachitta. (May 2016) The Impact of a Low Level of Agreement Among Reviewers in a Code Review Process https://www.researchgate.net/publication/303097906

As machine learning and AI driven solutions are growing stronger, automated code review tools have gone beyond syntax-based analysis. Modern AI powered tools, in particular Large Language Models (LLMs) can provide in context suggestions that aids in more intricate coding discussions in terms of accuracy, and efficiency (Chen et al., 2021). The next section explores in which ways these advancements made it possible for the integration of LLMs into the software engineering.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

III. LARGE LANGUAGE MODELS IN SOFTWARE ENGINEERING

3.1 Overview of LLMs and Their Capabilities

It is clear that large language models (LLMs) are definitely becoming transformative tools in software engineering; not only with NLP, but also with the solutions and reasoning in complex problems with software. OpenAI's Codex, DeepMind's AlphaCode, Google's PaLM: LLMs that have been trained on massive datasets of natural language and source code to understand, produce and optimize software code as well as with human proficiency (Chen et al., 2021). Specifically, transformer-based neural networks serve as deep learning architectures in these models to process extensive contextual information. The self-attention mechanisms within LLMs enable them to discover the relationships between sequence tokens which makes them exceptionally capable of processing code syntax as well as semantics and intents (Brown et al, 2020). The capability of LLMs allows them to tackle various tasks within software engineering through functions that complete code and refactor it and detect errors while also generating software documentation.

Software engineering benefits greatly from LLMs because they perform generalization between multiple programming languages at once. The multilingual competence of LLMs differs from rule-based systems because they gather their syntax rules automatically from datasets which contain code blocks composed in Python Java C++ JavaScript and Rust. The ability of LLMs to understand different programming languages makes them useful assets for developers who work across varied technology frameworks because they eliminate the need for specialist language knowledge and increase efficiency (Austin et al., 2021).

LLMs possess the capability to transform programs into understandable explanations which proves beneficial for developer education while also helping new staff members join teams and enhance team work dynamics. LMs provide contextual description that facilitates communication between programming levels enabling beginners to understand senior development methods while enhancing their programming abilities quickly (Sarsa et al., 2022).

3.2 How LLMs Process and Understand Code

Generation of code using LLMs occurs through distinct methods that differ fundamentally from current static and dynamic analysis procedures. The code interpretation and generation process of LLMs depends on tokenization with contextual embeddings and autoregressive generation to create syntactically and semantically accurate code.

- **Tokenization**: To start its processing of code the LLM first needs to break the textual information into separate tokens. A tokenized code structure features basic elements such as keywords and operators together with function names as well as variable name portions. Through tokenization LLMs attain the ability to transform complicated code frameworks into discretizable elements (Vaswani et al., 2017).
- **Contextual Embeddings**: After the text gets tokenized the model applies vector representations with high dimensions to each token. The search algorithms differ from traditional keyword-based approaches because LLMs implement contextual embeddings to evaluate the relationships between tokens within specific code lines. Through this capability LLMs can interpret multiple patterns and logic errors to determine the purpose of functions and classes (Chen et al., 2021).
- **Autoregressive Generation**: Most LLMs work through autoregressive text generation because they utilize the current sequence to predict upcoming tokens. Through this method LLMs can generate syntax-valid code sequences. The generation method of correct and functional code through LLMs comes with the undesirable side effect of false natural code outputs which seem legitimate to the human eye (Zhou et al., 2022).

Reinforcement learning together with fine-tuning methods enables LLMs to enhance their performance when dealing with practical software development projects. RLHF applies to improve OpenAI's ChatGPT and Codex models through human feedback which both enhances their ability to understand user priorities and decreases the likelihood of producing faulty or unsecure output (Ouyang et al., 2022). Static analysis tools based on traditional rule systems depend on human-generated heuristics so LLMs provide an adaptive learning capability to understand new coding guidelines. The capability to adapt makes such systems suitable for contemporary software development spaces that experience regular modifications to best practices and technologies (Jiang et al., 2022).

3.3 Applications of LLMs Beyond Code Review

The attention directed towards LLM applications for automated code review extends to multiple alternative uses. The software engineering field experiences major changes through LLM use because of these key impact zones:

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

3.3.1 Code Completion and Generation

Through their advancements LLMs have improved code completion tools by providing suggestions which extend past barebones keyword autocompleters. The standard autocompletes tools operating inside Integrated Development Environments (IDEs) make suggestions by referencing predefined syntax patterns. The LLM technology behind GitHub Copilot and TabNine creates whole functions by analyzing context to optimize code and present multiple code implementation possibilities (Chen et al., 2021).

Smart code creation eases developer workload so programmers concentrate on complex problem resolution instead of syntax. The framework increases developer productiveness by cutting out routine coding work which leads to accelerated prototyping together with enhanced iteration processes.

3.3.2 Bug Prediction and Automated Debugging

The second essential usage of LLMs is to forecast alongside stopping software bugs. Current bug detection tools scan program code while executing test cases for spotting possible issues. The code analysis functionality of LLMs activates pre-existent capabilities to identify logical problems and security flaws and runtime errors without specific test scenarios (Zhou et al., 2022). The most advanced LLM-based tools exhibit automated debugging abilities by both pinpointing problems while offering complete explanations alongside correction solutions. An LLM-based debugging assistant detects null pointer errors in Java programs while describing the error origin and offering corrected code examples to avoid it. Advanced developer performance along with improved software integrity becomes possible through this level of assistance.

3.3.3 Software Documentation and Code Explanation

The practice of maintaining current documentation has proven difficult for software engineering professionals over long periods. Most developers feel a strong aversion to documentation work which results in poor maintenance of project descriptions becoming outdated. Multitask LLMs analyze source code to generate systematic documentation that explains functions together with classes and Application Programming Interfaces. An LLM inserted into a development pipeline can, for instance, automatically generate docstrings, README files, and API documentation from the structure and intent of the code. This allows teams to lessen the weight of manual documentation work without sacrificing consistency between updated documentation and code (Sarsa et al., 2022).

3.3.4 Security Analysis and Code Hardening

The growing dangers posed by a cybersecurity vulnerability have led to investigation of automated security analysis using LLMs. LLMs analyze code for common types of security errors like SQL injection and XSS, as well as buffer overflows, to assist developers in identifying and resolving vulnerabilities before release. The security focus of LLMs can go up to some extent such that some security LLMs can even advise us with security coding pattern and best practices, which helps to raise overall software application's security posture (Jiang et al., 2022).

IV. ENHANCING CODE REVIEW WITH LARGE LANGUAGE MODELS

4.1 How LLMs Assist in Static and Dynamic Code Analysis

The large language models have drastically enhanced automated code analysis both on static and dynamic code review processes. Static code analysis means code review without execution in order to check syntax errors, style error or structural errors. Traditional static analysis tools like SonarQube and Checkstyle simply need to be tuned and haven't seen much new development for several years. However, LLMs are able to understand context, and hence are more able to detect logical flaws, redundancy, and other inefficiencies (Chen et al., 2021). Accelerating program testing via LLMs helps in test case generation, runtime error detection and performance optimization in dynamic code analysis. By analyzing the execution pattern, LLM can also find memory leak, concurrent issue, and unexpected behaviour. Even more advanced LLM powered tools can predict possible run time failures, a contribution to software reliability and robustness (Fan et al., 2023). With these capabilities LLMs are very helpful when they can be used to automate complex debugging process and thus reduce the work of developers.

4.2 Detecting Code Smells, Vulnerabilities, and Anti-Patterns

Structural weaknesses in code that indicate deeper problems—code smells—often result in poor maintainability and higher defect rates if they are not addressed. By learning patterns from huge codebases, LLMs can automatically identify common code smells like long methods, duplicated code, and excessive coupling. LLMs surpass traditional linters because they examine code context to explain problems in specific code segments and proposed modifications

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

according to Wang et al. (2023). Apart from it, LLMs are also important for identifying security vulnerabilities. SQL injections risks, improper input validations, insecure use of API, and buffer overflow issues are among the things they can detect. In contrast to conventional security scanners, LLMs have reasoning about security implications from historical data and best practice (Zhou et al., 2022). Due to their usefulness in SDLCs, namely secure software development lifecycles (SDLCs), this makes them especially useful. Among other things, LLM driven code review is key to detecting anti-patterns, bad coding practices that degrades software quality over time. Hardcoded credentials, deeply nested loops and inefficient recursion are examples of it. Sarsa et al. (2022) indicate that LLMs can analyze the context, understanding whether a pattern is not acceptable or useful and provide alternatives to improve code efficiency and maintainability.

4.3 Improving Code Readability and Maintainability Through LLM Recommendations

Collaborative software development always relies on code readability and maintainability. Poorly written code will slow onboarding, mess up debugging, and create technical debt. Third, LLMs can help with readability by suggesting good read that better variables names, great comments and clean formatting. Whereas simple auto formaters do have that, LLMs consider context and intent, and suggested improvements should cohere with best coding practices (Austin et al., 2021).

One major advantage of LLMs is that they can automatically document. The can explain complex functions, APIs, and class structures in a concise format but include the entire detail. This makes development easier for developers, and the way they work without maintaining unnecessary code and effort. Furthermore, LLM generated summaries are particularly useful in fast-moving large-scale projects, since they help teams quickly get up to speed with codebases (Sarsa et al., 2022). LLMs increase the long-term software quality by improving code readability and maintainability. However, they have the capacity to provide personalized recommendations based on requirements related with the projects, thus making them one of the most powerful tools for the modern development workflow. However, LLMs do contribute to improving code review, but their limitations and challenges — false positives, bias, interpretability issues etc. — should be also addressed in the following subsection.

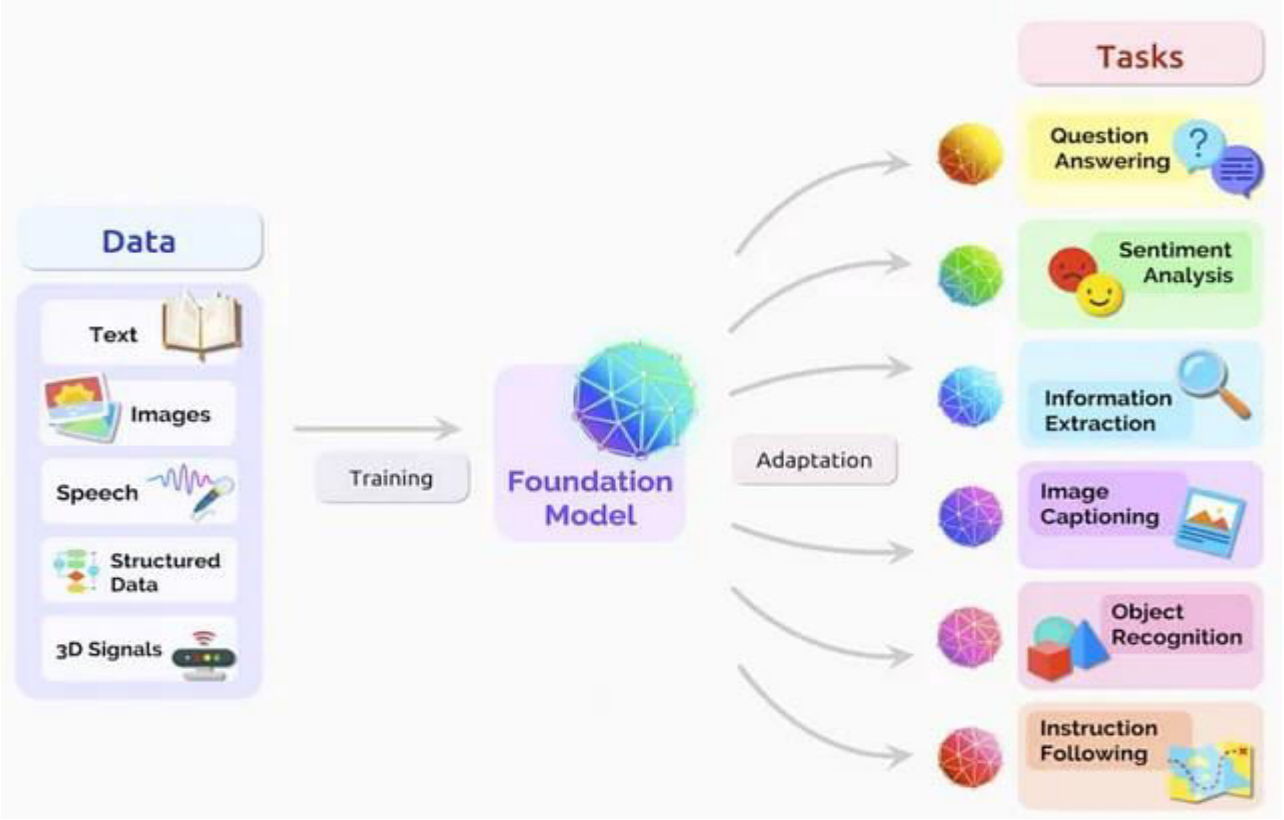

Figure 4: Understanding Large Language Models.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

Source: Bobur (Jul 27, 2023). LLM (Large Language Models) for better developer learning of your product. https://medium.com/@bumurzaqov2/llm-large-language-models-for-better-developer-learning-of-your-product-f353e7f49f06

V. IMPROVING SOFTWARE QUALITY THROUGH PREDICTIVE ANALYTICS

5.1 Predicting Defects Using LLM-Powered Analysis

Overall, the most valuable contribution that Large Language Models (LLMs) have made in software engineering is predicting software defects before they appear in production. However, all of these models assume historical bug data, code metrics, and rule-based systems, while being useful, often miss hidden correlation and coding patterns. By leveraging LLMs trained on large code repositories, defect prediction is improved for LLMs which assimilate patterns for syntax, logic structures outlook and bug histories in a codebase (Fan et al., 2023).

LLMs can use natural language processing (NLP) and deep learning skills to alert you of the possible defects in your code by indicating if there are ambiguous logic, redundant code blocks or incorrect function calls. For example, CodeT5+ and CodeGen have shown the power of pointing out the areas of code that are most likely to fail if we side by side comparisons of its patterns with the similar projects (Wang et al., 2023, Nijkamp et al., 2022). In addition, LLMs can give explanations as to why a particular segment is inclined to defects that developers can use to intervene. Defect prediction automation greatly diminishes the manual debugging time spent and allow the developers to proactively address errors rather than getting into a reactive bug fixing mode. It offers better predictability on overall software reliability & robustness, which in turn means reducing the number of post deployment failures and thereby improving an application's user experience.

5.2 Identifying Performance Bottlenecks and Optimization Suggestions

Slow execution times, improper memory usage and high CPU consumption are problems that degrade the software quality by adding performance bottlenecks. However, traditional profiling tools as well as static analyzers can help in identifying such issues, but they need manual interpretation and lack context awareness. With the help of LLMs, we can use code execution flow and function dependencies as well as compute efficiency to offer real time suggestions for optimization which has been demonstrated by (Austin et al., 2021).

These architectures are capable of detecting bad unoptimized loops, object creation and recursive calls where there is unnecessary potential for performance improvement, and so suggest refactored code which can improve performance. Continuing into structured LLMs deployed on automated pipelines, even advanced LLM can 'read' runtime logs and telemetry to predict slowdowns before they come to an end user (Meyer et al., 2023). An added feature of LLMs is being used in database query optimization so that SQL queries will be structured properly so that the execution time and the consumption of a resource will be reduced. In addition to detection, LLMs may suggest more effective algorithms or parallel processing, which increases system efficiency. This enables developers to speed up debugging and optimization of software without having to manually tune it. It is particularly valuable for large sized enterprise applications that can garner millions of dollars of loss due to performance bottlenecks.

5.3 Case Studies of LLM-Driven Software Quality Improvements

Through several real-world case, it is shown how the application of LLM driven analytics manifests into software quality. For instance, GitHub Copilot, an AI powered code assistant has contributed to decrease the programming flaws and enhance the productivity of developers. AI assisted coding tools help developers write cleaner and more efficient code, and as a result, reduce the bug fix work that is required post deployment by a large margin (Zhou et al., 2022).

For example, LLM integration in automated testing frameworks is another example. For example, companies such as Google and Microsoft have utilized AI driven generation of test cases via LLMs to better cover test cases (Fan et al., 2023). By performing so, this approach guarantees that the software goes through the toughest quality assurance checks before its deployment to drastically reduce the risk of crashes or a performance drop without notice. Beyond this, there have been uptakes of such LLM powered software refactoring tools within large scale open source projects so that they provide help to maintain existing codebases which are legacy in nature. LLMs reduce technical debt (McIntosh et al., 2016) by automating code restructuring and documentation, enabling developers to practice safe design and not constantly rewrite code, while developing and maintaining complex architectures. The fact that the predictive analytics powered by LLMs are not mere theoretical concept but have been and can be used in real world software development

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

can be seen in these case studies. LLMs are actively changing the shape of the future of software quality assurance by helping in defect prediction, performance optimization, automated refactoring.

VI. CHALLENGES AND LIMITATIONS OF LLM-BASED CODE REVIEW

However, LLMs, achieving great success in automated code review, present several challenges and limitations of their adoption. Accuracy and reliability are only a few concerns when it comes to accuracy and reliability. It is important for LLM driven code review systems to be both efficient and trustworthy, and addressing such challenges is required for that to happen.

6.1 Accuracy and Reliability Concerns

Accuracy and reliability of LLM based code review is one of the major concerns. Because LLMs are trained on masses of code, they don't always fully comprehend the meaning of a software project. Such misinterpretations, incorrect recommendations, or incomplete analysis can arise (Fan et al., 2023). As an example, an LLL may mark a piece of code as inefficient but not consider constraints that exist for this code, i.e., from the perspective of the project, it may not include constraints such as dependences with older versions or dependences from the domain, or optimizations only applicable in certain domains (Chen et al., 2021). Furthermore, complex business logic, where the logic is domain specific and the correctness of the code is not solely judged by syntax and structure, but more importantly by LLMs is something they are bad at. The second reliability problem is with LLMs being probabilistic — they don't follow strict rule-based logic but through learning patterns from the data, they produce responses. That is, rival queries can incite responses that are different, casting doubt on the trust developers have in AI-generated reviews (Meyer et al., 2023).

6.2 Handling False Positives and Negatives

Automated code analysis features high false positives and negatives. An LLM's false positive occurs when it flags correct code as defective and in turn causes wasted refactoring efforts. On the other hand, a false negative is when an LLM does not recognize an existing problem and, as a result, results into allowing bugs or security vulnerabilities to go undetected (McIntosh et al., 2016). As an example, LLMs trained with open sources may not be optimal for proprietary or specific programming frameworks, and thus their prediction will not be accurate as well. Clearly, this poses a serious problem in industries with high safety requirements such as healthcare and finance where they are extremely bad when a minor software error occurs (Singhal et al., 2022). Therefore, researchers have been exploring hybrid approaches to mitigate these risks, here LLM based reviews are combined with rule-based verification systems. But some systems permit developers to get LLMs on tip to give them extra ability to differentiate between real defects and fake alarms (Nijkamp et al., 2022).

6.3 Ethical Considerations and Biases in AI-Generated Reviews

Just like AI driven systems, in general (like LLMs), they are not immune to biases. For LLMs are trained on the existing codebases what they learn is the historical biases in software development practices. As such, for example, studies have shown that LLMs can project biases in code structure preferences, variable naming conventions, and even choose which programming language to use (Zhou et al., 2022). On the one hand, one ethical problem is that it reinforces bad coding practice. And if an LLM is trained on poor or decayed code, it may recommend poor solutions instead of refining the code quality (Thoppilan et al, 2022). Moreover, there is a risk that AI code reviews may favor some coding style over another or reduce diversity in the approaches programming languages are utilized. Another problem is the responsibility issue: if an LLM mistakes code, who's to blame? Who should bear the responsibility? The developer, the organization pushing the LLM out, the model developers? These are ethical questions that highlight the need of human oversight and governance of AI assisted code review.

6.4 Scalability and Computational Constraints

Largely, LLMs require abundant computational resources for them to perform well. Running LLMs on large scale codebases can be a very expensive compute operation, which may not be feasible for small teams and startups (Austin et al., 2021). It is especially expensive to train and fine tune large models like CodeT5+ or GPT-4 on GPUs, especially if their memory capacity is very high. Furthermore, it may slow down software development cycles (Meyer et al., 2023) by running real time LLM driven code reviews on continuous integration/continuous deployment (CI/CD) pipelines. Some organizations tackle these challenges with lighter weight alternatives including smaller LLM variants or hybrid approaches which combine LLM's with traditional static analysis tools. It also comes with serverless computing and cloud-based AI services rise that can reduce infrastructure burden to provide LLM driven reviews (Wang et al., 2023).

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

VII. FUTURE PROSPECTS AND RESEARCH DIRECTIONS

Large Language Models (LLMs) are only becoming more advanced, and they have a lot of promise as they relate to enhancing code review and software quality assurance. Advancements in LLM architectures capable of understanding code better are among the most promising areas of research. According to the current models, the syntactic comprehension and code generation are improved, however further architectures in future may dedicate more on semantic understanding, reasoning and contextual awareness. Incorporating GNNs and reinforcement learning could enable LLMs to better understand the software logic and execution patterns and result in much more precise defect detection and performance optimization. Also, putting together multi modal learning where the models take text descriptions and visual representation of code into consideration may help such code understanding systems better detect complex bugs and inefficiencies.

There is another key direction, the unification of LLMs with traditional static analysis tools, using the predictive power of AI and deterministic rule-based validation. Traditional tools are good at identifying well-formed problems, such as memory leaks, buffer overflows, type mismatches, but are not easily extensible to ever changing coding patterns. As LLM based pattern recognition / anomaly detection becomes a reality, modern code review pipelines can simultaneously improve precision and accommodate LLM context without getting overwhelmed by false positive and negative results. Additionally, integrating AI driven suggestions with formal verification tools will be valuable for improving security standard compliance of LLMs, making LLMs more suitable to critical software applications such as healthcare, finance, and cybersecurity.

It is also expected that the emergence of LLM-driven code review will concentrate on the future of human-AI collaboration to achieve hybrid review systems that combine humans and AI to improve software quality. LLMs cannot be used as replacement for human reviewers, rather they can be used as smart assistants to engineers, automatically making routine checks, making optimization suggestions and signaling the portions that require more manual inspection. Developing more and more transparent AI models is the trend that is more and more often focused on creating the explanations of recommendations that help the developer validate the AI generated insights and refine based on it. AI enabled code review systems cannot just accelerate the software development cycles, but they also enable developers to learn better and create higher quality software.

VIII. CONCLUSION

LLMs (Large Language Models) have massively shifted automated code review into an era of efficiency improvement, higher quality & maintainability edge for software. They help find bugs, improve efficiency and enhance code readability… that all helps the modern software development game and is good to have. They require addressing the reliability, with difficulties including false positives, biases and scalability. Although LLMs could substantially enhance the quality of software, human intervenor is necessary to nudge the roughish insights generated from AI. More advanced AI architectures and engagement with LLMs into existing review systems will greatly expand their influence in software development in future.

REFERENCES

1. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.
2. McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2016). An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering, 21, 2146-2189.
3. Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023, May). Large language models for software engineering: Survey and open problems. In 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE) (pp. 31-53). IEEE.
4. Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022, August). Automatic generation of programming exercises and code explanations using large language models. In Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1 (pp. 27-43).
5. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 9, September 2023 ||

| DOI:10.15680/IJIRSET.2023.1209004 |

6. Wu, T., Terry, M., & Cai, C. J. (2022, April). Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In Proceedings of the 2022 CHI conference on human factors in computing systems (pp. 1-22).
7. Wang, Y., Le, H., Gotmare, A. D., Bui, N. D., Li, J., & Hoi, S. C. (2023). Codet5+: Open code large language models for code understanding and generation. arXiv preprint arXiv:2305.07922.
8. Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022, November). Large language models are human-level prompt engineers. In The Eleventh International Conference on Learning Representations.
9. Jiang, E., Toh, E., Molina, A., Olson, K., Kayacik, C., Donsbach, A., ... & Terry, M. (2022, April). Discovering the syntax and strategies of natural language programming with generative language models. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (pp. 1-19).
10. Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H. T., ... & Le, Q. (2022). Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239.
11. Meyer, J. G., Urbanowicz, R. J., Martin, P. C., O'Connor, K., Li, R., Peng, P. C., ... & Moore, J. H. (2023). ChatGPT and large language models in academia: opportunities and challenges. BioData mining, 16(1), 20.
12. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., ... & Xiong, C. (2022). Codegen: An open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2203.13474.
13. Singhal, K., Azizi, S., Tu, T., Mahdavi, S. S., Wei, J., Chung, H. W., ... & Natarajan, V. (2022). Large lnguage models encode clinical knowledge. arXiv preprint arXiv:2212.13138.
14. McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2014, May). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In Proceedings of the 11th working conference on mining software repositories (pp. 192-201).
15. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35, 27730-27744.
16. Bobur (Jul 27, 2023). LLM (Large Language Models) for better developer learning of your product. https://medium.com/@bumurzaqov2/llm-large-language-models-for-better-developer-learning-of-your-product-f353e7f49f06
17. Toshiki Hirao, Akinori Ihara, Yuki Ueda, Passakorn Phannachitta. (May 2016)The Impact of a Low Level of Agreement Among Reviewers in a Code Review Process https://www.researchgate.net/publication/303097906
18. Shenwai D. S., (July 4, 2023). **What are Large Language Models (LLMs)? Applications and Types of LLMs**
19. Hiren Dhaduk (2019) Code quality: Its Importance in Custom Software Development. https://www.simform.com/blog/code-quality/

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor: 8.423

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details