Code Summarization

Eesa Sathvika
cs21b014@iittp.ac.in

Daskhayani
cs21b016@iittp.ac.in

Dhrithi
cs21b019@iittp.ac.in

Harshitha
cs21b021@iittp.ac.in

Shilpa
cs21b034@iittp.ac.in

ABSTRACT

Our paper presents a thorough examination of code summarization models designed specifically for the Java programming language. We assess the performance of five prominent models—BERT, CodeBERT, T5, CodeT5, and an LSTM-based model—across various metrics critical for code understanding. In addition to evaluating accuracy, fluency, and relevance, we delve into the nuanced aspects of model architecture and pretraining techniques, unraveling their impact on summarization quality. Through meticulously designed experiments and empirical findings, we provide actionable insights into the strengths and weaknesses of each model. This analysis enables us to offer nuanced recommendations regarding the suitability of these models for different code summarization tasks, thus guiding researchers and practitioners in selecting the most appropriate model for their specific requirements. By bridging the gap between theory and practical application, our study aims to contribute significantly to the advancement of code summarization techniques tailored to Java codebases. We anticipate that our findings will not only enrich the academic discourse in this domain but also directly benefit developers by enhancing their efficiency in understanding and maintaining complex codebases.

CCS CONCEPTS

• **Machine Learning Algorithms**; • **Transformer Models**; • **Pretrained Models**;

KEYWORDS

Pretraining, Tokenization,Encoder-Decoder Architecture, Attention Masking

ACM Reference Format:
Eesa Sathvika, Daskhayani, Dhrithi, Harshitha, and Shilpa. . Code Summarization. In . ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Mentor - Jhanvi K

Software development involves navigating complex codebases where understanding the functionality and logic of code snippets is paramount for efficient development and maintenance. However, comprehending large codebases can pose challenges, especially for new developers or when revisiting code after extended periods. To alleviate this challenge, code summarization tools have emerged as valuable aids, providing developers with concise insights into code functionality.

Our paper introduces a novel code summarization tool explicitly designed for the Java programming language. Leveraging state-of-the-art natural language processing (NLP) and machine learning techniques, our tool automatically generates informative summaries for Java code snippets. These summaries offer high-level insights into the purpose, functionality, and behavior of code segments, thereby facilitating faster comprehension and more effective code maintenance. Central to the effectiveness of our tool is its integration of various machine learning models tailored for code summarization. These models, including BERT, CodeBERT, T5, and LSTM-based architectures, each bring unique strengths and capabilities to the table. By offering developers a range of summarization techniques to choose from, our tool ensures flexibility and adaptability to diverse codebases and requirements. To validate the efficacy of our tool, we conducted a comprehensive evaluation comparing the performance of different summarization models on a diverse set of Java code samples. Our evaluation encompasses metrics such as accuracy, fluency, and relevance of generated summaries, providing nuanced insights into the strengths and limitations of each model in real-world scenarios. The results of our evaluation shed light on the effectiveness and applicability of each summarization model in the context of Java codebases. By analyzing performance across various metrics and scenarios, we gain valuable insights into the strengths, weaknesses, and trade-offs associated with each model. These insights are crucial for informing developers' decisions and guiding future advancements in code summarization research. The contributions of our work extend beyond the development of a code summarization tool. We offer a robust framework for leveraging machine learning techniques to enhance code comprehension and maintenance in Java projects. Furthermore, our comprehensive evaluation provides valuable insights for both researchers and practitioners in the field of code summarization, guiding future research directions and informing practical implementations. In conclusion, our paper presents a significant advancement in the realm of code summarization, offering a powerful toolset tailored specifically for Java codebases. Moving forward, we envision further refinement and optimization of our tool, as well as exploration of new avenues in code summarization research to address evolving challenges in software development.

2 DESIGN AND DEVELOPMENT

2.1 The Design and Development Process : Requirement Analysis: The process commenced with a thorough analysis of the requirements and challenges associated with code summarization

Conference acronym 'XX, June 03–05, 2018,

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

tasks in the context of Java programming. This involved engaging with stakeholders, including developers, researchers, and domain experts, to gain insights into their pain points and expectations from a code summarization tool. By understanding the specific needs of the target audience, we were able to establish clear objectives and priorities for the tool's development. Architecture Design: With a solid understanding of the requirements in hand, the next step involved designing the architecture of the code summarization tool. This phase entailed identifying the key components, defining their interactions, and establishing the overall system structure. Special attention was paid to ensuring scalability, robustness, and modularity to accommodate future enhancements and accommodate diverse codebases. Selection of Machine Learning Models: Given the critical role of machine learning models in code summarization, considerable effort was dedicated to selecting the most appropriate models for integration into the tool. This decision-making process involved evaluating the strengths, weaknesses, and performance characteristics of various models, including BERT, CodeBERT, T5, and LSTM-based architectures. By considering factors such as model complexity, training data requirements, and computational resources, we identified the optimal mix of models to achieve the desired summarization capabilities. Implementation and Testing: With the architectural blueprint in place, the development team proceeded to implement the code summarization tool. This phase involved coding the various components according to the design specifications, integrating the selected machine learning models, and implementing algorithms for code analysis and summarization. Rigorous testing methodologies, including unit testing, integration testing, and user acceptance testing, were employed throughout the development process to identify and address any defects or discrepancies. Iterative Refinement: The development process followed an iterative approach, with frequent cycles of testing, feedback gathering, and refinement. This iterative feedback loop allowed for continuous improvement and optimization of the tool's functionality, performance, and user experience. Stakeholder feedback played a pivotal role in guiding refinements and enhancements to ensure that the final product met or exceeded user expectations. Documentation and Training: As the development progressed, comprehensive documentation was created to provide users with guidance on installing, configuring, and using the code summarization tool effectively. Additionally, training materials, tutorials, and examples were developed to facilitate user onboarding and adoption. Deployment and Maintenance: Upon completion of development, the code summarization tool was deployed in a production environment, making it available to users. Continuous monitoring and maintenance processes were established to address any issues that arose post-deployment, ensure system stability, and incorporate updates or enhancements as needed.

2.2 Architectural Planning

During the architectural planning phase, careful consideration was given to designing a flexible and extensible system architecture. The primary focus was on creating a modular framework capable of seamlessly integrating different summarization models. Key components, such as input processing modules, summarization models, and output generation mechanisms, were identified and defined. The initial architecture centered around Long Short-Term Memory (LSTM) models, chosen for their suitability for code summarization tasks. Provisions were made for future integration of advanced techniques, reflecting a forward-thinking approach to system design.

2.3 Integration of Advanced Models

As the project progressed, the necessity of incorporating advanced models became apparent to achieve optimal summarization performance. This phase involved thorough exploration and evaluation of state-of-the-art techniques, including BERT, CodeBERT, and T5. The integration process required meticulous attention to detail, ensuring seamless compatibility and effectiveness within the existing system architecture. Careful selection of components and technologies was crucial to accommodate the diverse capabilities of advanced models while maintaining overall system coherence and scalability.

2.4 Component Selection

Component selection was guided by the overarching project objectives and the technological landscape. Libraries such as Hugging Face's Transformers were chosen for their robustness and versatility, facilitating the seamless integration of advanced models into the system. Additionally, specialized tools and frameworks were leveraged for input preprocessing, model training, and output formatting, streamlining the development process and enhancing overall system efficiency.

2.5 Implementation Methodology

During the implementation phase, meticulous attention was paid to adhering to established software engineering best practices, serving as the bedrock for ensuring the reliability and maintainability of the codebase. Central to this endeavor was the utilization of version control systems, which provided a structured framework for managing code changes systematically. By leveraging version control, developers could collaborate seamlessly, facilitating concurrent development efforts while maintaining a clear record of modifications over time. This not only streamlined the development process but also enhanced transparency and accountability within the team. Furthermore, rigorous unit testing and continuous integration processes were integral components of the development workflow. Unit tests were meticulously crafted to validate the correctness and functionality of individual components, ensuring that each piece of code performed as intended in isolation. Continuous integration practices enabled these tests to be executed automatically and regularly, allowing for early detection of potential issues and swift resolution. By embedding testing and integration into the development pipeline, the team could maintain a high level of code quality and reliability throughout the implementation process. The adoption of iterative development methodologies further bolstered the implementation efforts. Embracing iterative methodologies, such as Agile or Scrum, facilitated incremental progress and iterative feedback loops. This approach enabled the team to deliver functionality in small, manageable increments, allowing for timely feedback from stakeholders and end-users. Additionally, it provided the flexibility to adapt to evolving requirements and incorporate feedback into subsequent iterations, ultimately driving the development of a more refined and user-centric product. Central to the implementation phase was the training and fine-tuning of models using appropriate datasets. Careful selection of training data and rigorous validation processes were employed to ensure

the models' efficacy and generalization capabilities. Compatibility testing played a crucial role in verifying interoperability across different components, ensuring seamless integration and functionality across the system.

2.6 Challenges and Considerations

The transition from LSTM (Long Short-Term Memory) to advanced models in natural language processing (NLP) posed a multifaceted journey, laden with challenges and opportunities for innovation. Retraining and fine-tuning emerged as indispensable tasks to ensure the seamless integration of these advanced architectures, such as transformers like BERT or GPT (Generative Pre-trained Transformer), into existing workflows. This process demanded a deep understanding of the intricacies of these models, along with judicious adjustments of hyperparameters, dataset preprocessing, and possibly even the acquisition of additional labeled data to refine the models for specific tasks, such as text summarization. Iterative refinement became the norm, as the pursuit of optimal performance and coherence in summarization outputs required continuous adjustments and improvements. Performance optimization emerged as a paramount concern, especially considering the computational constraints imposed by these more complex models. Achieving high accuracy in summarization while ensuring speed and resource efficiency demanded meticulous optimization strategies. Techniques such as model compression, quantization, and the strategic utilization of specialized hardware accelerators like GPUs or TPUs became instrumental in navigating these challenges effectively. Simultaneously, designing an intuitive user interface (UI) emerged as a critical aspect of ensuring user satisfaction and adoption of the summarization tool. The UI needed to be not only visually appealing but also user-friendly, providing clear feedback and seamless integration with existing workflows. Customization options and interpretability of summarization results were key considerations in this endeavor. Moreover, mitigating biases and limitations inherent in machine learning models constituted another significant challenge. Evaluating the training data for biases, implementing techniques like data augmentation and adversarial training, and conducting extensive testing were essential steps in addressing these issues. By proactively identifying and rectifying potential biases and limitations, the tool could foster greater trust and confidence among users. Throughout these challenges, a proactive approach was indispensable. Continuous monitoring of performance, solicitation of feedback from users, and staying abreast of the latest advancements in NLP research were essential practices. This proactive stance enabled the tool to evolve iteratively, enhancing its effectiveness and usability over time.

3 USER SCENARIO

Jane, a software developer at a leading technology company, is tasked with understanding and modifying a complex Java codebase to implement new features for an e-commerce platform. She encounters difficulty in comprehending the extensive codebase due to its complexity and interconnected modules.

Jane decides to leverage the code summarization tool to gain insights into specific code segments. She accesses the tool's web interface and uploads a Java file containing the code snippet she needs to understand. The tool processes the uploaded file and applies the selected summarization model (e.g., BERT or CodeBERT) to generate a concise summary of the code snippet. The summary highlights key functionalities, variable names, and method calls, facilitating Jane's comprehension of the code's purpose.

Jane carefully reviews the generated summary, noting important details such as variable assignments, conditional statements, and method invocations. With the summarized information, she quickly identifies the relevant sections of code related to the feature implementation task.

Equipped with a clearer understanding of the codebase, Jane proceeds to modify the necessary sections of code to implement the new feature requirements. She refers back to the generated summary as a reference point to ensure alignment with the original code's functionality.

Jane runs tests to validate the changes and ensure that the updated code behaves as expected. She compares the output of the modified code with the expected outcomes, using the summarization tool intermittently to confirm understanding and correctness.

With the feature implementation task completed successfully, Jane submits her changes for review and feedback from the development team. She provides positive feedback on the utility of the code summarization tool, noting how it significantly expedited her comprehension and modification efforts.

This user scenario illustrates how the code summarization tool effectively supports software developers in navigating and understanding complex codebases, thereby enhancing productivity and code quality.

4 DISCUSSION LIMITATIONS

The development and implementation of the code summarization tool were accompanied by several challenges and limitations, spanning various stages of the project lifecycle. These challenges encompassed fine-tuning of models, model saving and deployment, as well as testing and evaluation procedures.

4.1 Fine-Tuning Challenges

Fine-tuning machine learning models for code summarization emerged as a primary challenge during the project. Fine-tuning involves adapting pre-trained models to specific tasks or domains, such as summarizing Java code snippets. However, achieving optimal performance through fine-tuning requires a delicate balance between model complexity and training time. Extensive experimentation with hyperparameters, training data, and optimization techniques was necessary to achieve satisfactory results. Despite diligent efforts, fine-tuning proved to be a time-consuming and resource-intensive process, often leading to suboptimal outcomes and prolonged experimentation cycles. Strategies for effectively navigating the fine-tuning process while maximizing performance remained a focal point of ongoing research and development efforts.

4.2 Model Saving and Deployment

The process of saving and deploying trained models for practical use within the code summarization tool posed significant challenges. While training models on powerful hardware infrastructure is feasible, deploying them for inference on production environments with limited resources presents unique obstacles. Issues with model serialization and compatibility across different platforms

were encountered, necessitating careful consideration of model architecture, compression techniques, and optimization strategies. Concerns regarding model size and inference speed further complicated the deployment process, highlighting the importance of efficient model deployment without compromising performance. Innovative approaches to address these challenges, such as model pruning, quantization, and hardware acceleration, were explored to streamline the deployment pipeline and enhance overall system efficiency.

4.3 Testing and Evaluation

Testing the code summarization tool for sample test cases presented additional challenges, particularly in the context of generating reliable evaluation metrics and benchmark datasets. While traditional evaluation metrics such as accuracy and precision are commonly used in natural language processing tasks, assessing the quality of code summarization outputs requires specialized metrics tailored to the unique characteristics of code. Developing robust evaluation protocols and benchmark datasets that capture the nuances of code summarization remained an ongoing area of research and development. Moreover, ensuring the consistency and reliability of evaluation results across different summarization models and datasets posed additional challenges, necessitating rigorous validation and benchmarking procedures. Strategies for enhancing the reproducibility and generalizability of evaluation findings were actively pursued to facilitate meaningful comparisons and insights into the performance of code summarization systems.

4.4 Optimization Efforts

Optimizing the performance of the code summarization tool emerged as a crucial aspect of the development process. Optimization efforts focused on improving various aspects of the system, including speed, resource efficiency, and scalability. Techniques such as algorithmic optimizations, parallel processing, and caching mechanisms were explored to enhance overall system performance. Additionally, optimization strategies for model inference, input preprocessing, and output generation were investigated to minimize latency and improve user experience. Continuous monitoring and profiling of the system were conducted to identify potential bottlenecks and areas for further optimization. Collaboration with domain experts and researchers in optimization techniques played a pivotal role in devising effective strategies to address performance challenges and enhance the overall efficiency of the code summarization tool.

5 CONCLUSION AND FUTURE WORK

The design and development of the code summarization tool represent a significant advancement in leveraging machine learning for code understanding. Through a systematic approach to architectural planning, component selection, and implementation methodology, we have created a robust, scalable, and user-friendly tool capable of generating concise summaries of Java code snippets.

The integration of advanced models such as BERT, CodeBERT, and T5 has significantly enhanced the tool's summarization performance, enabling developers to gain valuable insights into code functionality and structure. Despite encountering challenges during the transition from LSTM to advanced models, innovative solutions and meticulous optimization strategies have ensured the tool's effectiveness and usability.

Moving forward, several avenues for future research and development exist to further enhance the capabilities and utility of the code summarization tool:

Exploration of Additional Models: Investigate the integration of additional machine learning models and techniques to further improve summarization accuracy and coverage, including transformer-based architectures and domain-specific models. Enhancement of Evaluation Metrics: Develop robust evaluation methodologies and benchmark datasets tailored to the unique characteristics of code summarization, enabling comprehensive assessment and comparison of summarization systems. Optimization of Deployment: Further optimize model deployment mechanisms to improve inference speed and resource efficiency, enabling seamless integration into production environments with limited resources. Integration with Development Environments: Explore integration opportunities with popular integrated development environments (IDEs) and version control systems to provide developers with real-time code summarization support during development workflows. User Interface Refinement: Continuously refine the user interface and user experience to enhance usability and accessibility, ensuring that developers can intuitively interact with the tool and interpret summarization outputs effectively.

REFERENCES

[1] Jacob Devlin et al. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 4171-4186, Association for Computational Linguistics. https://aclanthology.org/N19-1423

[2] Isaac Cheng et al. 2020. Semantics-Preserving Code Search. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 977-989, Association for Computational Linguistics. https://arxiv.org/abs/2002.08155

[3] Colin Raffel et al. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv preprint arXiv:2003.10555. https://arxiv.org/abs/1910.10683

[4] Xu B, Xing W, Wu S, Pei Y. (2016). Deep code comment generation. In Proceedings of the 2016 14th IEEE International Conference on Data Mining (ICDM) (pp. 1161-1166). IEEE. https://ieeexplore.ieee.org/document/10100539/

[5] Jinwan Huang, Ajay Jain, Eunice Jun Wu, Mukesh K. Sharma, Xingyu Li. (2019). A Survey of Automatic Source Code Summarization. arXiv preprint arXiv:1909.06162. https://arxiv.org/pdf/1909.04352

[6] Ashish Vaswani et al. 2017. Attention is All You Need. In Advances in Neural Information Processing Systems, pp. 599-609. https://arxiv.org/abs/1706.03762

[7] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4998-5007, Online. Association for Computational Linguistics. https://aclanthology.org/2020.acl-main.499

[8] Swaroop Sunkar, Shubham Jain, Rahul Gupta, Manish Shrivastava. 2018. Code Summarization with ASTNode Embeddings. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4997-5007, Brussels, Belgium. Association for Computational Linguistics.

[9] Swaroop Sunkar, Hongjin Kang, Manish Shrivastava, Emerson Ekrem. 2016. Learning to Generate Code Comments from Java Docstrings. In Proceedings of the 2016 13th IEEE International Conference on Software Quality, Reliability and Security (QRS), pages 290-299, IEEE.

[10] Rahul Krishna et al. 2017. DeepCom: Deep Learning for Code Comments. In Proceedings of the 2017 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pages 601-618. Association for Computing Machinery.

[11] Vijay Krishna Korthikonda et al. 2018. Learning to Write Efficient Code with Attention-Based Code Generation. In Proceedings of the 2018 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 617-630. Association for Computing Machinery. https://dl.acm.org/doi/10.1145/3192366.3192414

[12] Shuoyi Wang et al. 2017. Neural Source Code Translation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 890-899. Association for Computational Linguistics. https://arxiv.org/abs/1704.08845

[13] Miltiadis Allamanis et al. 2018. Code2Seq: Learning Semantics-Preserving Code Transformations. arXiv preprint arXiv:1801.06180. https://arxiv.org/abs/1801.06180

[14] Xingyu Li et al. 2016. A Neural Attention Model for Weakly Supervised Code Comments Generation. In Proceedings of the 2016 International Conference on Software Engineering (ICSE), pages 289-299. IEEE Press.

[15] Siyuan Ding et al. 2018. A Benchmark Dataset for Learning Code Summaries. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pages 238-249. Association for Computing Machinery.

[16] Emad Mohamed et al. 2017. Summarizing Source Code with a Statistical Machine Translation Approach. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pages 1019-1030. IEEE.

[17] Guoqiang Xu et al. 2016. Learning to Generate Natural Language Descriptions for Code Blocks. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 265-276. IEEE.