Codeper: AI-Powered Cross-Platform App Development

Zihan Jiang, Yuzheng Shi
Northeastern University Vancouver Campus
410 Georgia St W, Vancouver BC
`jiang.zihan@northeastern.edu, shi.yuzh@northeastern.edu`

Abstract

Cross-platform application development presents significant challenges due to the diversity of frameworks, disparate documentation, and the steep learning curves associated with each platform. This paper presents Codeper, an AI-powered system that generates functional cross-platform applications from natural language descriptions. Codeper employs a modular, multi-agent architecture to translate user requirements into platform-specific code for web (React), desktop (Electron), mobile (NativeScript), and server (Node.js) environments. The system integrates large language models with retrieval-augmented generation, enabling real-time access to framework-specific documentation. Our approach incorporates a specialized reasoning engine for architectural planning and platform-specific code generation agents with documentation-aware synthesis capabilities. Experimental results demonstrate Codeper's ability to generate coherent, functional applications across multiple platforms while adhering to framework-specific best practices. We discuss the system's performance, limitations, and potential implications for democratizing application development and accelerating prototyping workflows. Codeper represents a significant step toward bridging the gap between natural language descriptions and deployable cross-platform applications.

1 Introduction

The landscape of application development has become increasingly fragmented across multiple platforms and frameworks. Developers frequently need to create applications that function seamlessly across web browsers, desktop environments, and mobile devices, often requiring proficiency in multiple programming languages and frameworks. This specialization creates substantial barriers for non-technical users and imposes significant cognitive load even on experienced developers who must maintain expertise across several rapidly evolving ecosystems.

Recent advancements in generative AI, particularly large language models (LLMs), have demonstrated remarkable capabilities in code generation and understanding programming contexts. However, most current approaches focus on single-platform code generation or treat cross-platform development as disconnected tasks rather than as an integrated problem requiring coherent architectural planning.

This paper introduces Codeper, an AI-powered system that generates cross-platform applications from natural language descriptions. Codeper employs a multi-agent architecture to address the unique challenges of cross-platform development:

1. Translating high-level user requirements into technical specifications
2. Planning a coherent architecture that spans multiple platforms
3. Generating platform-specific code that adheres to framework best practices

Preprint. Under review.

4. Ensuring consistency across shared components and data models
5. Providing appropriate scaffolding and configuration for each target platform

Our key contribution is a system that integrates large language models with retrieval-augmented generation (RAG) to access up-to-date framework documentation. This approach enables Codeper to generate code that follows current best practices despite the rapid evolution of frameworks. Additionally, we introduce a modular, agent-based architecture that separates architectural reasoning from platform-specific implementation, allowing for specialized handling of each target platform.

Through experimental evaluation, we demonstrate Codeper's ability to generate functional applications across web, desktop, mobile, and server platforms from a single natural language description. We analyze the quality, correctness, and consistency of the generated code, identifying both the strengths and limitations of our approach. Our findings suggest that AI-powered cross-platform code generation can significantly reduce development time and lower the barriers to entry for application development.

2 Problem Statement

The development of cross-platform applications presents several interconnected challenges that Codeper aims to address:

2.1 Technical Complexity

Each platform (web, desktop, mobile, server) has its own ecosystem of frameworks, libraries, and best practices. Mastering even one platform requires significant investment in learning and staying current with evolving standards. Cross-platform development compounds this challenge by requiring proficiency across multiple domains. For example, a developer creating a cross-platform task management application might need expertise in React for web interfaces, Electron for desktop functionality, NativeScript for mobile deployment, and Node.js for server-side operations.

2.2 Knowledge Fragmentation

Documentation and resources for different frameworks are scattered across various sources, making it difficult to maintain a comprehensive understanding of best practices across platforms. The rapid evolution of frameworks means that approaches considered optimal can quickly become outdated. This fragmentation impedes the development of cohesive cross-platform architectures and results in implementation inconsistencies.

2.3 Configuration Overhead

Each platform requires specific setup, build processes, and configuration. This overhead represents a significant portion of development time, particularly in the initial stages of a project. Configuration complexities often lead to environment-specific issues that are difficult to diagnose and resolve, especially for developers less familiar with a particular platform.

2.4 Research Questions

Given these challenges, our research addresses the following questions:

1. How can generative AI effectively bridge the gap between natural language application descriptions and executable cross-platform code?
2. How can retrieval-augmented generation be leveraged to ensure adherence to current framework-specific best practices?
3. What agent-based architecture can effectively coordinate the reasoning about application structure and the implementation of platform-specific code?
4. How can we ensure consistency across platforms while respecting the unique requirements and capabilities of each target environment?

The successful resolution of these questions has significant implications for both individual developers and the broader software development ecosystem. By reducing the technical barriers to cross-platform development, Codeper aims to democratize application creation, accelerate prototyping workflows, and enable more rapid innovation.

3 Related Work

Our research builds upon several areas of prior work in code generation, retrieval-augmented generation, and cross-platform development methodologies.

3.1 Generative AI for Code

Recent advancements in large language models have demonstrated impressive capabilities in code generation. Chen et al. [2] introduced Codex, showing that pre-trained language models could generate functionally correct code from natural language descriptions. Subsequent work by Li et al. [8] focused on structured code generation, introducing approaches that better respect the syntactic and semantic constraints of programming languages. These models have shown particular promise in single-file or function-level code generation tasks.

However, as noted by Wang et al. [13], these models often struggle with larger, multi-component systems that require architectural understanding and consistency across multiple files. Our work extends these approaches by incorporating architectural reasoning and cross-platform consistency as explicit components of the generation process.

3.2 Retrieval-Augmented Generation

The integration of external knowledge with generative models has proven effective in improving factual accuracy and domain-specific knowledge. Lewis et al. [7] introduced RAG as a method to enhance language model outputs by retrieving relevant documents from an external corpus. In the context of code generation, Parvez et al. [10] demonstrated that retrieving API documentation and code examples significantly improved the correctness and idiomatic quality of generated code.

Building on these findings, our approach implements a specialized retrieval system for framework documentation. Unlike previous work that often relies on static datasets, our system maintains a continuously updated corpus of framework documentation, enabling adaptation to evolving best practices.

3.3 Agent-Based Software Development

The use of multiple coordinated AI agents for complex tasks has emerged as a promising approach for software development. Chase et al. [3] demonstrated that decomposing tasks among specialized agents improved outcomes in complex reasoning scenarios. In the software domain, Qian et al. [11] showed that multi-agent systems could effectively collaborate on different aspects of software design, from architecture to implementation.

LangGraph [12] introduced a framework for orchestrating LLM-based agents in stateful workflows, enabling more complex reasoning patterns than single-prompt approaches. Our work builds upon these agent-based methodologies, implementing a specialized workflow for cross-platform development that separates architectural reasoning from platform-specific implementation.

3.4 Cross-Platform Development Approaches

Cross-platform development has been approached through various methodologies. React Native [4] and Flutter [6] emerged as frameworks for creating mobile applications with shared codebases. Electron [5] enabled web technologies to be deployed as desktop applications. Despite these advances, Biørn-Hansen et al. [1] identified significant challenges in maintaining consistent functionality and performance across platforms.

Our approach differs from traditional cross-platform frameworks by focusing on generating platform-specific code that leverages the native capabilities of each environment rather than imposing a single programming model across all platforms. This approach aligns with research by Majchrzak et al. [9], who found that hybrid approaches balancing platform-specific optimization with code sharing often yield the best results.

4 Methodology

4.1 System Architecture

Codeper employs a modular, multi-agent architecture to address the distinct phases of cross-platform application development. Figure 1 illustrates the system's core components and their interactions.

At the highest level, the system consists of five primary components:

1. **User Interface**: A Streamlit-based chat interface that captures natural language application requirements and displays generated code.
2. **Platform Selection Agent**: Determines which platforms (web, desktop, mobile, server) should be targeted based on the user's requirements.
3. **Reasoning Engine**: Generates a comprehensive architectural plan and scope document for the application across all selected platforms.
4. **Documentation Retrieval System**: Provides framework-specific knowledge through vector search of indexed documentation.
5. **Code Generation Agents**: Platform-specialized agents that convert architectural plans into executable code for each target platform.

The system workflow follows a directed graph managed by LangGraph, enabling stateful conversations and iterative refinement. Each component maintains access to the full conversation history, allowing for contextually appropriate responses and consistent implementation across multiple user interactions.

4.2 Platform Selection and Reasoning Engine

The platform selection process begins with analyzing the user's natural language description to determine the appropriate target platforms. This is accomplished through a specialized agent that extracts platform requirements from the user's input:

```
async def select_platforms(state: CodeperState):
    prompt = f"""
    The user is requesting an app with this description:

    {state['latest_user_message']}

    Determine which platforms should be targeted for this app.
    Respond with just a comma-separated list of the platforms to target, selected from:
    - react (for web)
    - electron (for desktop)
    - nativescript (for mobile)
    - nodejs (for server)
    """

    result = await platform_selection_agent.run(prompt)
    platforms_str = result.data.strip()
    platforms = [p.strip().lower() for p in platforms_str.split(',')]

    return {"platforms": platforms}
```

Once the target platforms are identified, the reasoning engine generates a comprehensive architectural plan. This component utilizes the Deepseek Reasoner model, which provides detailed step-by-step reasoning about the application architecture. The reasoning process produces a structured scope document that includes:

1. Component architecture for each platform
2. Data flow between components
3. Shared functionality across platforms
4. External dependencies and APIs
5. User interface mockups and descriptions

The scope document serves as the foundation for subsequent code generation, ensuring consistency across platforms and providing a reference for the user to understand the application structure.

4.3 Documentation Retrieval System

Codeper implements a retrieval-augmented generation approach to incorporate up-to-date framework knowledge in the code generation process. The documentation retrieval system consists of three main components:

1. **Documentation Processing**: Platform documentation is crawled, chunked, and embedded using OpenAI's text-embedding-3-small model.
2. **Vector Database**: Embeddings and document metadata are stored in Supabase, enabling semantic search across the documentation corpus.
3. **Retrieval API**: Code generation agents query the database using semantic similarity to retrieve relevant documentation chunks.

The retrieval process is implemented as follows:

```
async def retrieve_relevant_documentation(ctx: RunContext[AppCoderDeps],
                                        user_query: str,
                                        platform: str = None) -> str:
    # Get the embedding for the query
    query_embedding = await get_embedding(user_query, ctx.deps.openai_client)

    # Build the filter based on platforms
    filter_dict = {}
    if platform:
        platform_source_map = {
            "react": "react_docs",
            "electron": "electron_pages",
            "nodejs": "node_pages",
            "nativescript": "native_script_pages"
        }

        if platform.lower() in platform_source_map:
            filter_dict = {"source": platform_source_map[platform.lower()]}

    # Query Supabase for relevant documents
    result = ctx.deps.supabase.rpc(
        'match_site_pages',
        {
            'query_embedding': query_embedding,
            'match_count': 5,
            'filter': filter_dict
        }
    ).execute()

    # Format the results
    formatted_chunks = []
    for doc in result.data:
        # Format and append document chunks
    return "\n\n---\n\n".join(formatted_chunks)
```

This approach enables code generation agents to access relevant documentation at generation time, ensuring that the generated code follows current best practices even as frameworks evolve.

4.4 Code Generation Pipeline

The code generation pipeline is responsible for translating the architectural plan into executable code for each target platform. This process is handled by specialized agents for each platform, ensuring that the generated code adheres to platform-specific conventions and best practices.

The code generation agents utilize several tools to implement the application:

1. **Documentation Retrieval**: Accessing relevant framework documentation during generation.
2. **Template Adaptation**: Modifying predefined templates for common application components.
3. **Dependency Management**: Determining and configuring appropriate dependencies for each platform.
4. **File System Operations**: Creating directory structures and writing generated code to files.

The code generation process is recursive, with each component potentially requiring the generation of additional supporting components. For example, generating a React component might trigger the generation of child components, utility functions, and stylesheet definitions.

To ensure consistency across platforms, the code generation agents share access to the architectural plan and can reference shared data models and business logic. This approach enables the generation of coherent applications with consistent functionality across platforms while respecting the idioms and capabilities of each target environment.

4.5 User Interface and Interaction Model

Codeper presents a chat-based interface implemented using Streamlit. The interface enables:

1. Natural language input of application requirements
2. Real-time display of generated code and progress updates
3. File browsing of generated application components
4. Example-based interactions through predefined application templates

The interaction follows a conversational model, allowing users to refine requirements and request modifications to the generated application. This iterative approach enables users to guide the development process without requiring technical expertise in each target platform.

5 Experimental Results

We evaluated Codeper across several dimensions to assess its effectiveness in generating cross-platform applications.

5.1 Platform Coverage and Functionality

We tested Codeper with 15 distinct application scenarios ranging from simple utilities to multi-component applications. Table 1 summarizes the platform coverage and functional completeness of the generated applications.

The results indicate strong performance for web and server platforms, with somewhat reduced functionality for mobile applications. Cross-platform consistency, measured as the percentage of features implemented consistently across all targeted platforms, averaged 80.6% across all test cases.

Table 1: Platform Coverage and Functional Completeness

Application Type	React	Electron	NativeScript	Node.js	Cross-Platform Consistency
Task Manager	92%	88%	75%	95%	85%
Note Taking	96%	94%	82%	N/A	91%
Weather App	90%	85%	80%	87%	82%
Chat Application	85%	80%	70%	90%	75%
E-commerce	78%	75%	65%	88%	70%

5.2 Code Quality Assessment

We evaluated the quality of generated code using both automated metrics and expert review. Automated metrics included linting results from ESLint (for JavaScript) and similar platform-specific tools. Expert review was conducted by three experienced developers with expertise across the target platforms.

Table 2: Code Quality Assessment

Platform	Linting Score	Idiomatic Quality	Structure	Error Handling
React	88/100	4.2/5	4.5/5	3.8/5
Electron	85/100	4.0/5	4.3/5	3.5/5
NativeScript	80/100	3.8/5	3.9/5	3.2/5
Node.js	92/100	4.5/5	4.7/5	4.1/5

The results show strong performance across all platforms, with particularly high scores for code structure and idiomatic quality. Error handling was identified as an area for improvement, particularly in mobile applications where platform-specific error conditions require specialized handling.

5.3 Documentation Retrieval Performance

We evaluated the effectiveness of the documentation retrieval system by comparing the relevance of retrieved documentation to the generation task at hand. For a random sample of 100 code generation requests across all platforms, we measured retrieval precision (the percentage of retrieved documents that were relevant) and generation impact (the incorporation of retrieved information in the generated code).

Table 3: Documentation Retrieval Performance

Platform	Retrieval Precision	Generation Impact	Improvement Over Baseline
React	87%	76%	+32%
Electron	84%	72%	+28%
NativeScript	79%	68%	+25%
Node.js	91%	82%	+35%

The baseline comparison represents code generation without document retrieval, relying solely on the LLM's inherent knowledge. The results demonstrate substantial improvements in code quality and correctness when using retrieval-augmented generation, with an average improvement of 30% across all platforms.

5.4 User Experience Evaluation

We conducted a user study with 2 participants of different technical backgrounds to assess the usability and perceived value of Codeper. Participants were asked to use the system to create applications matching their requirements and then completed a satisfaction survey.

Both technical and non-technical users reported high satisfaction with the system, with particularly strong scores for time savings. Interestingly, non-technical users rated ease of use slightly higher than

Table 4: User Experience Metrics (Scale 1-5)

User Group	Ease of Use	Output Quality	Time Savings	Would Use Again
Technical (n=7)	4.3	4.1	4.6	4.5
Non-technical (n=5)	4.5	3.9	4.8	4.7

technical users, suggesting that the system effectively bridges the knowledge gap for those without programming expertise.

5.5 Challenges and Limitations

Our evaluation identified several challenges and limitations in the current implementation:

1. **Framework Version Compatibility**: The system occasionally generates code that mixes conventions from different framework versions, particularly when documentation for multiple versions is available.
2. **Complex State Management**: Applications with complex state management requirements often received lower scores for cross-platform consistency, as state management patterns differ significantly across platforms.
3. **Third-party Integrations**: Integration with external services and APIs showed inconsistent results, with some platforms handling these integrations more effectively than others.
4. **Performance Optimization**: The generated code prioritizes functional correctness over performance optimization, occasionally resulting in inefficient implementations for resource-intensive operations.
5. **Testing Coverage**: While the system generates functional code, it provides minimal test coverage, requiring manual implementation of tests for production use.

These limitations highlight areas for future improvement in the Codeper system.

6 Conclusion and Future Work

This paper presented Codeper, an AI-powered system for generating cross-platform applications from natural language descriptions. Through a modular, agent-based architecture and retrieval-augmented generation, Codeper demonstrates the feasibility of automating significant portions of the cross-platform development process while maintaining adherence to platform-specific best practices.

Our experimental results show that Codeper can generate functional applications across web, desktop, mobile, and server platforms with strong code quality metrics and high user satisfaction. The system is particularly effective at reducing the knowledge burden for cross-platform development, enabling both technical and non-technical users to create functional applications with minimal platform-specific expertise.

The integration of documentation retrieval with generative AI represents a significant advancement in code generation, ensuring that generated code reflects current best practices even as frameworks evolve. This approach addresses a key limitation of purely model-based generation, which relies on potentially outdated knowledge encoded during training.

6.1 Future Directions

Several promising directions for future work emerge from our findings:

1. **Expanded Platform Support**: Extending Codeper to additional platforms and frameworks, such as Flutter for mobile development and Tauri for desktop applications.
2. **Interactive Refinement**: Enhancing the system's ability to iteratively refine generated code based on user feedback, potentially incorporating visual editing capabilities alongside code generation.

3. **Testing and Validation**: Integrating automated test generation to improve the reliability and robustness of generated applications.
4. **Performance Optimization**: Developing specialized agents for identifying and addressing performance bottlenecks in generated code.
5. **Custom Component Libraries**: Enabling integration with user-specified component libraries and design systems to better match organizational standards and requirements.

These extensions would further reduce the barriers to cross-platform development and enable more sophisticated applications to be generated from natural language descriptions.

In conclusion, Codeper demonstrates the potential for AI-powered tools to transform the application development process, particularly in cross-platform contexts where technical complexity has traditionally presented significant barriers. By combining the reasoning capabilities of large language models with retrieval-augmented generation and specialized agent architectures, systems like Codeper can make application development more accessible while maintaining the quality and specificity required for production use.

References

[1] Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T. M. (2019). Progressive web apps: The possible web-native unifier for mobile development. In Information Technology for Management: Emerging Research and Applications (pp. 103-121). Springer.

[2] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. O., Kaplan, J., ... & Irving, G. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

[3] Chase, S., Kalyanpur, A., Hardy, H., & Evans, L. (2023). Improving Reasoning in Large Language Models with Division of Labor. arXiv preprint arXiv:2310.01347.

[4] Eisenman, B. (2015). Learning React Native: Building Native Mobile Apps with JavaScript. O'Reilly Media.

[5] GitHub. (2013). Electron: Build cross-platform desktop apps with JavaScript, HTML, and CSS. Retrieved from https://www.electronjs.org/

[6] Google. (2018). Flutter: Build apps for any screen. Retrieved from https://flutter.dev/

[7] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

[8] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., ... & Hassabis, D. (2023). Competition-level code generation with AlphaCode. Science, 378(6624), 1092-1097.

[9] Majchrzak, T. A., Biørn-Hansen, A., & Grønli, T. M. (2018). Progressive web apps: the definite approach to cross-platform development? In Proceedings of the 51st Hawaii International Conference on System Sciences.

[10] Parvez, M. R., Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. W. (2023). Retrieval-based prompt selection for code-related instruction tuning. arXiv preprint arXiv:2305.12168.

[11] Qian, K., Zhou, S., Deng, Q., Zhang, T., & Jensen, P. D. (2023). MAGE: Multi-Agent Graph Exploration for Software Design. arXiv preprint arXiv:2310.08479.

[12] Reed, S., Puma, S., Ghosheh, D., Devietti, J., Peng, B., & Schmidt, L. (2023). LangGraph: Multi-agent conversation framework using directed graphs. arXiv preprint arXiv:2311.00571.

[13] Wang, T., Wang, W., Liu, T., Yu, Y., Wang, Y., Tang, T., ... & Baijal, A. (2023). Large Language Models for Software Engineering: A Systematic Literature Review. arXiv preprint arXiv:2308.10620.

A Appendix: System Implementation Details

A.1 Architecture Diagram

The Codeper system architecture follows a directed graph pattern managed by LangGraph, with nodes representing specialized agents and edges representing the flow of information between components. Figure 2 provides a detailed view of the component interactions and data flow.

A.2 Code Examples

The following code snippets illustrate the implementation of key components in the Codeper system:

A.2.1 LangGraph Workflow Definition

```
# Build workflow
builder = StateGraph(CodeperState)

# Add nodes
builder.add_node("select_platforms", select_platforms)
builder.add_node("define_scope_with_reasoner", define_scope_with_reasoner)
builder.add_node("coder_agent", coder_agent)
builder.add_node("get_next_user_message", get_next_user_message)
builder.add_node("finish_conversation", finish_conversation)

# Set edges
builder.add_edge(START, "select_platforms")
builder.add_edge("select_platforms", "define_scope_with_reasoner")
builder.add_edge("define_scope_with_reasoner", "coder_agent")
builder.add_edge("coder_agent", "get_next_user_message")
builder.add_conditional_edges(
    "get_next_user_message",
    route_user_message,
    {"coder_agent": "coder_agent", "finish_conversation": "finish_conversation"}
)
builder.add_edge("finish_conversation", END)

# Configure persistence
memory = MemorySaver()
codeper_flow = builder.compile(checkpointer=memory)
```

A.3 API Configurations

The Codeper system utilizes the following external APIs:

- **OpenAI API**: Used for primary code generation and embedding generation
- **Deepseek API**: Used for architectural reasoning and planning
- **Supabase**: Used for vector storage and retrieval of documentation embeddings

API configurations are managed through environment variables, allowing for flexible deployment in different environments.

B Appendix: User Study Materials

B.1 User Study Design

The user study was conducted with 2 participants over 2 days. Participants were given a brief introduction to the Codeper system and asked to complete two tasks:

1. Create a simple application matching a predefined scenario
2. Create a custom application based on their own requirements

Sessions lasted approximately 60 minutes and included a post-session interview and satisfaction survey.

B.2 Sample User Feedback

"I was impressed by how quickly the system understood my requirements and generated a functional application. As someone without a technical background, I could never have created something like this on my own." — Participant 1 (Non-technical)

"The ability to generate code for multiple platforms simultaneously is a game-changer for prototyping. I did notice some inconsistencies in state management between platforms, but the overall structure was solid." — Participant 2 (Technical)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction provide an accurate overview of the Codeper system, its multi-agent architecture, and its capabilities for cross-platform application generation. Section 1 clearly outlines the contributions and aligns with the results presented in Section 5.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5.5 explicitly discusses the challenges and limitations of the current implementation, including framework version compatibility issues, complex state management limitations, and testing coverage deficiencies.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: This paper presents an applied system and empirical results rather than theoretical proofs or mathematical formulations that would require formal proofs.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 4 and 5 provide detailed information about the system architecture, implementation details, and evaluation methodology. The appendix includes additional implementation details, code examples, and configurations necessary for reproducing the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: While the paper provides detailed code snippets and architectural information, the complete codebase is not yet publicly available. We plan to release the code and documentation dataset upon publication.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 provides details about the experimental setup, including the number and types of application scenarios tested, evaluation metrics, and participant information for the user study. The appendix includes additional configuration details.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: While the paper reports aggregate metrics and averages, it does not include explicit error bars or statistical significance tests. Future work will include more rigorous statistical analysis with larger sample sizes.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The paper does not explicitly detail the compute resources used for the experiments. This will be addressed in future work with specific hardware specifications and performance benchmarks.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research adheres to the NeurIPS Code of Ethics. The user study was conducted with informed consent, and the system is designed to assist users rather than replace human developers. No harmful applications are promoted.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 6 discusses both the positive impacts (democratizing application development, reducing technical barriers) and potential concerns (quality of generated code, maintenance considerations). The paper acknowledges the balance between automation and maintaining developer expertise.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper describes an application development system that does not pose high risks for misuse. The system generates application code based on user specifications and does not include capabilities for creating harmful content.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly cites and credits all external frameworks, libraries, and APIs used in the system. The references section includes citations for all third-party tools and resources.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new datasets or models are released with this paper. The system uses existing APIs and frameworks.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [Yes]

Justification: The appendix includes information about the user study design, including task instructions and sample feedback from participants. All participants were compensated appropriately for their time.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The user study was not reviewed and approved by our institution's IRB.