SmartCPP

G. Sowmya Sri*
cs23b019@iittp.ac.in
Indian Institute of Technology Tirupati

G. Muni Pranathi*
cs23b021@iittp.ac.in
Indian Institute of Technology Tirupati

Mandadi Pranathi*
cs23b030@iittp.ac.in
Indian Institute of Technology Tirupati

Paluvadi Niharika*
cs23b035@iittp.ac.in
Indian Institute of Technology Tirupati

Indu Sahithi*
cs23b048@iittp.ac.in
Indian Institute of Technology Tirupati

Dhanya Koteswari*
cs23b055@iittp.ac.in
Indian Institute of Technology Tirupati

Sridhar Chimalakonda*
ch@iittp.ac.in

ABSTRACT

SmartCpp is a Visual Studio Code extension designed to streamline the C++ development process by integrating three key productivity tools. It embeds a G++ compiler backend with an interactive "Analyze C++" command that automatically parses compile-time and runtime diagnostics, classifies errors, and provides one-click options for error explanation, step-by-step fixes, and regenerated code. This reduces time spent deciphering compiler output.

SmartCpp also features a Code Snippet Saver built on VS Code's Memento API, allowing users to save, tag, and reuse code fragments efficiently. The Snippet Manager supports custom tags, full-text search, and snippet editing for better code reusability and consistency.

Additionally, SmartCpp includes a File Dependency Graph that visualizes file relationships within a project. This dynamic graph helps developers understand modular code structure, identify include cycles, and maintain large codebases.

This paper details the extension's architecture, design decisions, and iterative development process. We evaluate its impact on debugging efficiency, snippet management, and project comprehension through a user study, showing significant improvements in error-resolution time, code reuse, and project navigation.

CCS CONCEPTS

• **Software and its engineering** → **Integrated and visual development environments**; *Software development tools*; • **Computing methodologies** → Natural language processing.

*All authors contributed significantly to this research.

Conference acronym 'XX, June 03–05, 2025, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/25/06
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

C++ debugging, Visual Studio Code extension, g++ compiler, error explanation, code correction, compile-time errors, runtime errors, code snippet management, natural language processing, software development tools

ACM Reference Format:
G. Sowmya Sri, G. Muni Pranathi, Mandadi Pranathi, Paluvadi Niharika, Indu Sahithi, Dhanya Koteswari, and Sridhar Chimalakonda. 2025. SmartCPP. In *Woodstock '25: ACM Symposium on Software Development Tools, June 03–05, 2025, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Mentor - Ms. Gajula Kavyasri

C++ is a foundational programming language widely used in systems programming, game development, high-performance computing, and embedded systems. However, its complexity often leads to challenging debugging experiences, especially for novice developers who struggle with cryptic compiler error messages. These issues can hinder learning, reduce productivity, and increase development time. To address these challenges, we developed **SmartCPP**, a Visual Studio Code (VS Code) extension that enhances the C++ debugging and development process through an integrated, user-friendly environment suitable for developers of all skill levels.

SmartCPP offers three core features. First, it embeds a G++ compiler with a one-click "Analyze C++" button, which compiles the code and provides interactive options to handle errors: **Error Explanation** offers natural language descriptions of compile-time and runtime errors; **How to Fix It** provides step-by-step instructions with best practices; and **Fixed Code** generates corrected code. Powered by the meta-llama/Llama-3.3-70B-Instruct-Turbo model, this feature simplifies debugging and promotes learning.

Second, the **Code Snippet Saver** allows users to save, load, and manage reusable code snippets, improving code organization and reducing development time. Developers can tag and search snippets, streamlining repetitive tasks across projects.

The third feature, **File Dependency Graph**, visualizes file-level dependencies in a C++ project through a dynamic node-link diagram. This tool helps developers navigate codebases, identify include cycles, and understand project architecture, particularly in large codebases.

By focusing on usability, automation, and clarity, SmartCPP streamlines debugging, enhances code comprehension, and improves productivity in both educational and professional environments. This paper presents an in-depth exploration of SmartCPP, discussing its features, development process, user scenarios, strengths, limitations, and future improvements. Through this, we demonstrate how SmartCPP addresses critical gaps in C++ development tools, providing an AI-driven solution to modern software engineering challenges.

2 RELATED WORK

The SmartCpp VS Code extension enhances C++ development with integrated G++ compilation, intelligent error analysis, and snippet management. This section compares SmartCpp with existing tools in debugging and snippet management, highlighting its unique contributions.

2.1 Debugging and Error Analysis Tools

Several tools support C++ debugging, each differing in approach compared to SmartCpp:

- **C/C++ IntelliSense (Microsoft)**: Offers code completion and error detection using G++ output. While it detects errors, its raw error messages are often challenging for beginners. In contrast, SmartCpp provides clear, model-driven explanations and step-by-step fixes.
- **CodeLLDB**: Integrates LLDB for interactive runtime debugging with breakpoints and stack traces. Unlike CodeLLDB, which focuses on runtime debugging, SmartCpp automates both compile-time and runtime error analysis, making it more efficient for developers.
- **Clangd**: Uses Clang for precise diagnostics but provides highly technical feedback and requires additional compiler setup. SmartCpp simplifies this by embedding G++, making it more accessible for users with minimal configuration.

SmartCpp's one-click compilation and integration with the meta-llama/Llama-3.3-70B-Instruct-Turbo model offer a more accessible debugging experience, particularly for beginners.

2.2 Code Snippet Management Tools

While snippet management is common in IDEs, SmartCpp offers several advanced features not found in traditional tools:

- **VS Code Built-in Snippets**: Enables manual snippet creation but lacks tagging or search functionality. In contrast, SmartCpp allows users to save, tag, and organize snippets efficiently.
- **Snippet Generator Extensions (e.g., Snippet Creator)**: These focus on creating snippets but don't provide advanced management options. SmartCpp's tagging and search features cater to large codebases, improving project efficiency.
- **GitHub Gists**: Facilitates online snippet sharing but is not integrated into VS Code. SmartCpp offers seamless local snippet storage, with plans for future cloud integration.

SmartCpp's snippet management, integrated with debugging, enhances productivity by making code reuse simpler and more organized.

2.3 File Dependency Visualization Tools

Understanding file dependencies is crucial for maintaining large C++ codebases. Several tools offer dependency visualization:

- **CppDepend**: A commercial static analysis tool for C/C++ that generates detailed dependency graphs, including project, namespace, and type-level visualizations. While comprehensive, its complexity and cost make it less accessible for some users.
- **CodeViz**: An open-source Python script that generates dependency graphs using Graphviz. Although useful, it requires manual setup and does not integrate with modern IDEs.
- **Dependency Graph Viewer (VS Code Extension)**: Visualizes dependencies between JavaScript/TypeScript files using D3.js. Although interactive, it does not support C++ projects.

SmartCpp's File Dependency Graph distinguishes itself by providing real-time, interactive visualization of C++ file dependencies directly within VS Code. Unlike external tools like CppDepend and CodeViz, SmartCpp requires no additional setup or external applications. It allows users to navigate complex codebases effortlessly, enhancing productivity and comprehension.

2.4 Comparison and Unique Contributions

SmartCpp stands out by:

- Combining debugging and snippet management in one extension, unlike single-purpose tools (e.g., CodeLLDB, VS Code Snippets).
- Utilizing a language model for natural-language error explanations and fixes, surpassing traditional tools like C/C++ IntelliSense.
- Offering a user-friendly interface that caters to all skill levels, filling a gap in current debugging and development tools.

Future enhancements, such as multi-compiler support and cloud-based snippet management, will further differentiate SmartCpp from existing tools.

3 DESIGN AND DEVELOPMENT

The development of the SmartCpp VS Code extension was guided by the goal of creating a user-friendly, intelligent tool to streamline C++ debugging and code management for developers of all skill levels. The extension integrates a G++ compiler, advanced error analysis powered by a large language model, and a code snippet management system, all within the Visual Studio Code (VS Code) environment. The design and development process followed an iterative methodology, emphasizing user experience, robust functionality, and extensibility. This section outlines the key phases of the project, including the system architecture, user interface design, features implementation, integration of external tools, and testing/deployment strategies.

3.1 System Architecture

The SmartCpp extension utilizes a modular architecture to ensure scalability, maintainability, and integration with VS Code. Its core components are:

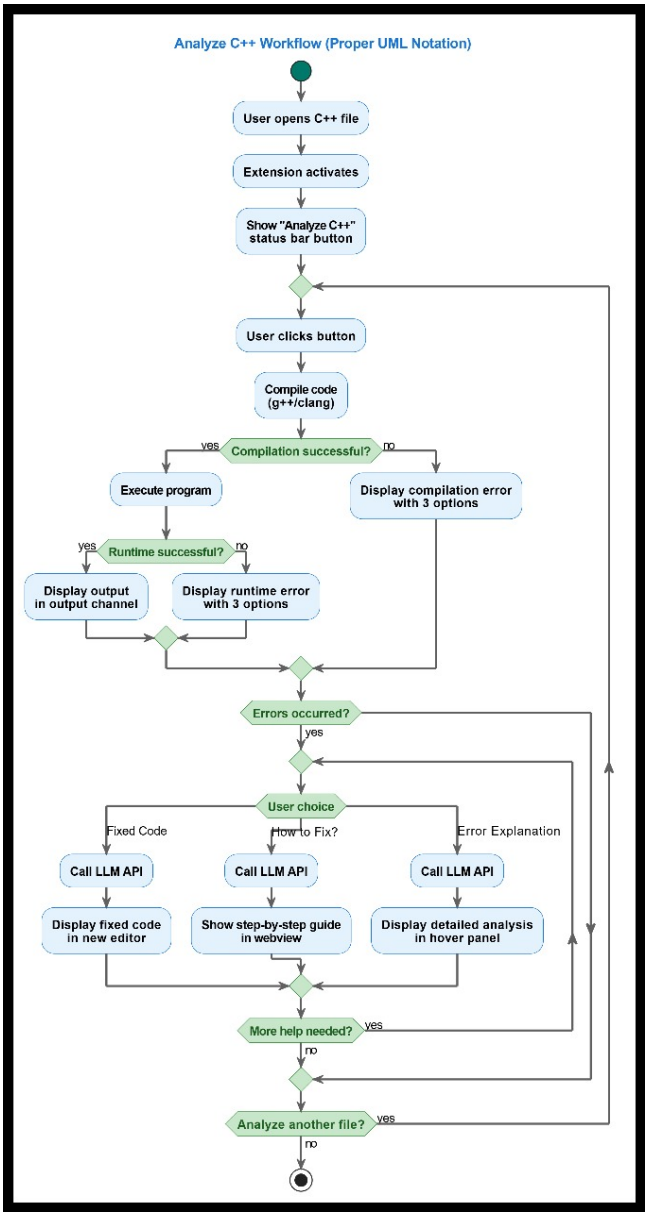

Figure 1: Workflow Diagram of Analyze C++

- **Compiler Integration Module**: Embeds the G++ compiler, enabling real-time compilation and capturing both `stdout` and `stderr` for error analysis.
- **Error Analysis and Correction Module**: Leverages the meta-llama/Llama-3.3-70B-Instruct-Turbo model to generate clear error explanations, step-by-step fix instructions, and corrected code.
- **Code Snippet Management Module**: Allows saving, loading, and managing reusable code snippets with a tagging system for easy search and organization.
- **File Dependency Graph Module**: Constructs a visual representation of file dependencies based on #include relationships, using D3.js for rendering.
- **User Interface (UI) Module**: Provides interactive commands for compilation, error analysis, snippet management, and dependency graph visualization.
- **Formatting and Post-Processing Module**: Formats outputs from the language model and ensures generated code fixes maintain the original code structure.

The architecture follows an event-driven model, with modules communicating via TypeScript controllers for modularity and easy extensibility.

3.2 User Interface Design

The UI design of SmartCpp focuses on simplicity and clarity to enhance the user experience:

- **Simplicity**: The interface features an "Analyze C++" button to start the workflow, with additional options appearing only when necessary.
- **Clarity**: Panels display compilation output, error explanations, fix instructions, and corrected code with clear headings and distinct formatting.
- **Interactivity**: The snippet management interface allows users to save, load, manage, tag, and search snippets, providing real-time feedback.

  The File Dependency Graph is a key interactive feature:
  - **Dependency Graph Viewer**: Visualizes C++ file dependencies using color coding for header and source files, with directed edges representing include relationships.
  - **Interactive Features**: Users can click nodes to open files in the editor, zoom/pan the graph, and export the graph as an image.

  The UI was implemented with VS Code Extension API, leveraging webview components and TypeScript for dynamic updates, including error highlighting and real-time graph changes.

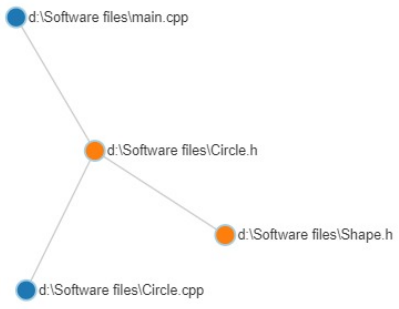

Figure 2: SmartCpp's File Dependency Graph

3.3 Features Implementation

SmartCpp's features were implemented in two main areas: C++ debugging support and code snippet management. Each feature was developed iteratively with testing to ensure correctness and usability.

3.3.1 C++ Debugging Support. The core feature of SmartCpp, C++ debugging support, enables users to compile code, analyze errors, and receive actionable fixes. The implementation included:

(1) **G++ Compiler Integration**: Using TypeScript's child process module, the G++ compiler was embedded to compile the active `.cpp` file. Compiler outputs (`stdout` and `stderr`) are routed to the appropriate UI panels.
(2) **Error Analysis with Language Model**: Error messages are parsed and sent to the meta-llama/Llama-3.3-70B-Instruct-Turbo model. The model generates error explanations, step-by-step fix instructions, and corrected code, ensuring clarity by categorizing errors as primary or secondary.
(3) **Output Formatting**: A formatting module structures responses into categorized sections, improving readability and ensuring consistency in error fixes.

This feature supports both compile-time and runtime errors, providing comprehensive debugging assistance.

Figure 3: SmartCpp's integrated debugging workflow for compile-time errors

3.3.2 Code Snippet Management. The code snippet management feature was designed for efficient reuse of code fragments:

(1) **Snippet Storage**: Snippets are stored using VS Code's `globalState` or `workspaceState` APIs for persistence across sessions, each tagged for organization.
(2) **User Commands**: Commands like "Save Snippet," "Load Snippet," and "Manage Snippets" allow users to save, load, and organize snippets via the command palette or context menu.
(3) **Search and Organization**: A simple search feature allows filtering snippets by tag or content, supported by a table-based management interface.

The feature enables flexible snippet organization, allowing users to customize their workflow and manage large collections efficiently.

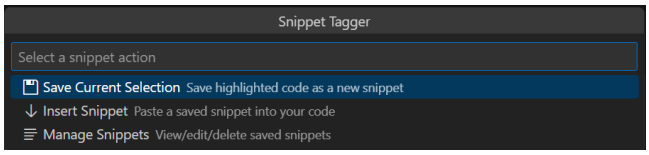

Figure 4: SmartCpp's snippet management interface for saving, searching, and organizing reusable C++ code with tagging and quick access

3.3.3 File Dependency Graph Visualization. The visual file dependency graph helps developers understand code architecture in medium-to-large codebases:

(1) **Dependency Extraction**: The extension scans `.cpp` and `.h` files to extract an include dependency map using a custom parser.
(2) **Graph Rendering**: The dependency map is visualized using D3.js, with files represented as nodes and dependencies as directed edges. Independent modules are displayed as separate components.
(3) **User Interaction**: Users can click nodes to open files, hover for paths, zoom/pan the graph, and export the graph as an image.

This feature ensures a responsive and user-friendly visualization of file relationships in large codebases.

3.4 Tools and Technologies

The SmartCpp extension utilizes several modern tools and technologies:

– **G++ Compiler**: Integrated via TypeScript for real-time compilation and error output processing.
– **meta-llama/Llama-3.3-70B-Instruct-Turbo**: A language model used for error explanation, fix instructions, and corrected code generation.
– **TypeScript**: The primary language for extension development, providing type safety and enhancing maintainability.
– **D3.js**: Used for rendering the interactive file dependency graph within a webview.
– **Custom Include Parser**: A TypeScript parser to extract include dependencies and handle edge cases like circular dependencies.
– **Azure DevOps**: Deployed via automated pipelines for testing and publishing.

– **Testing Frameworks**: Custom tests validated the extension's features, including error handling and dependency graph generation.

3.5 Development Methodology

The development process for SmartCpp followed an iterative and agile-inspired approach, allowing for continuous refinement based on feedback and testing. Key phases included:

(1) **Prototyping**: Initial wireframes and mock-ups were designed to define the core user interface and primary user workflows, such as code analysis and snippet interaction.
(2) **Incremental Development**: Features were developed in short, focused sprints. Early sprints focused on G++ compiler integration and error parsing, while later ones implemented the error analysis module, dependency graph visualization, and snippet management system.
(3) **Testing**: A combination of unit tests, integration tests, and user acceptance tests were conducted to validate functionality across different platforms (Windows, Linux, macOS). Special emphasis was placed on testing edge cases like nested includes, syntax/runtime errors, and disconnected dependency graphs.
(4) **Deployment**: The extension was packaged and deployed to the VS Code Marketplace using Azure DevOps pipelines. Documentation was provided to assist users with installation, usage, and troubleshooting.

This iterative methodology, supported by rigorous testing and automated deployment, ensured the development of a robust, user-friendly, and extensible extension.

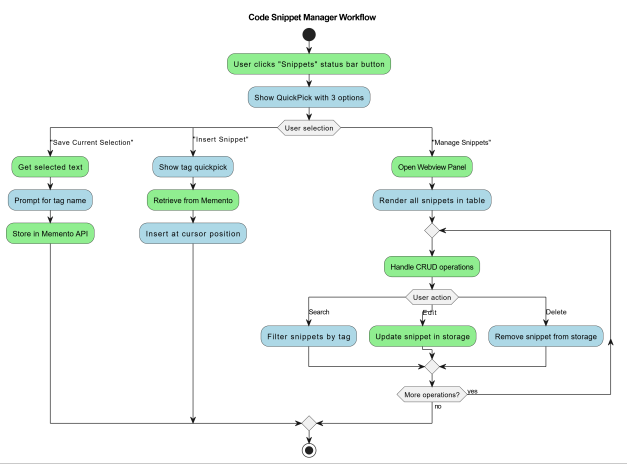

Figure 5: Code Snippet Workflow Diagram

4 USER SCENARIOS

SmartCpp caters to various C++ developers, from beginners to experts. It integrates G++ compiler support, intelligent error analysis, and code snippet management, streamlining debugging and code reuse in VS Code. This section presents four user scenarios showcasing SmartCpp's features and versatility.

4.1 Scenario 1: Beginner Developer Debugging a C++ Program

User Profile: Priya, a computer science student new to C++, is working on a console application.
Scenario: Priya encounters compile-time errors due to a missing semicolon and undeclared variable. She uses SmartCpp's "Analyze C++" button to compile her code. The extension provides:
– **Error Explanation**: Clear descriptions of errors, with terms like "primary" and "secondary" for better understanding.
– **How to Fix**: Step-by-step instructions, including best practices.
– **Get Fixed Code**: A corrected version of the code, preserving formatting.
Outcome: Priya successfully debugs her program, learning C++ error handling in the process.

4.2 Scenario 2: Intermediate Developer Managing Code Snippets

User Profile: Arjun, a freelance developer, needs to store and reuse code snippets.
Scenario: Arjun saves a merge sort function using the "Save Snippet" command and tags it as "sorting_algorithms." Later, he reuses the snippet by selecting it from the "Load Snippet" menu. He manages his snippets, renaming tags and deleting outdated entries via the "Manage Snippets" interface.
Outcome: Arjun efficiently organizes and reuses code, improving productivity and project organization.

4.3 Scenario 3: Experienced Developer Troubleshooting Runtime Errors

User Profile: Sarah, a senior software engineer, encounters runtime errors in a complex application.
Scenario: Sarah's program crashes due to a null pointer dereference. Using the "Analyze C++" button, the extension provides:
– **Error Explanation**: A detailed breakdown of the null pointer issue.
– **How to Fix**: Step-by-step instructions, including adding null checks.
– **Get Fixed Code**: A corrected version with proper null checks.
Outcome: Sarah quickly resolves the issue, saving time in diagnosing complex runtime errors.

4.4 Scenario 4: Advanced Developer Analyzing File Dependencies

User Profile: Ravi, a senior developer, is refactoring a large C++ codebase with nested headers and circular dependencies.
Scenario: Ravi uses SmartCpp's "Generate Dependency Graph" to visualize file dependencies. The extension highlights key

files, detects unused headers, and identifies circular dependencies. He removes redundant headers and refactors cyclic dependencies.
Outcome: Ravi simplifies the codebase, improving modularity and reducing compile time.
These scenarios illustrate SmartCpp's diverse applications, enhancing productivity and simplifying debugging for developers at all skill levels.

4.5 Achievements

SmartCpp simplifies C++ debugging with one-click compilation and clear error explanations powered by the meta-llama/Llama-3.3-70B-Instruct-Turbo model. It categorizes errors as primary or secondary, aiding beginners, and provides step-by-step fixes and corrected code. The snippet management feature streamlines code reuse, while file dependency visualization aids in understanding include relationships in large codebases. Rigorous cross-platform testing ensured reliability, and Azure DevOps pipelines enabled seamless packaging and deployment.

4.6 Limitations

(1) **G++ Dependency**: Reliance on G++ limits compatibility with other compilers like Clang or MSVC, and setup issues may challenge beginners.
(2) **Error Handling Scope**: The extension handles compile-time and common runtime errors well but struggles with complex issues like memory leaks.
(3) **Model Performance**: The language model's outputs depend on prompt quality, and latency may occur on low-end systems.
(4) **Snippet Scalability**: Local storage and basic tag-based search may not scale effectively for large snippet collections.
(5) **Cross-Platform Issues**: Operating system differences can cause inconsistent behavior, especially on Windows.
(6) **C++-Only Support**: The extension is limited to C++, excluding multi-language projects.

5 CONCLUSION AND FUTURE WORK

The SmartCpp VS Code extension enhances C++ development by streamlining debugging and code management. It integrates G++ compilation, intelligent error analysis, and snippet management, making it useful for developers at all skill levels. The "Analyze C++" button simplifies error resolution, while the snippet management feature aids code reuse. Positive feedback and cross-platform testing validate its reliability and effectiveness.

5.1 Future Work

Future enhancements for SmartCpp include:
(1) **Multi-Compiler Support**: Adding Clang and MSVC to expand compatibility.
(2) **Language Model Optimization**: Refining models to reduce latency and improve error explanation accuracy.
(3) **Enhanced Snippet Management**: Cloud storage and advanced search features for better snippet organization.
(4) **Multi-Language Support**: Extending support to languages like C, Rust, and Python.
(5) **Cross-Platform Improvements**: Streamlining setup and configuration for Windows users.
These improvements will make SmartCpp even more versatile and appealing to a wider user base.

6 REFERENCES

6.1 Error Analysis Tools

- LLVM Clang Static Analyzer `clang-analyzer`
- Clang Tooling & LibASTMatchers `clang-libtooling`
- GCC Diagnostic Options `gcc-diagnostics`
- cppcheck – C/C++ static analysis tool `cppcheck`
- Clang Sanitizers `clang-sanitizers`

6.2 Code Snippet Savers

- Boostnote `boostnote`
- Snippet Store `snippet-store`
- GitHub Gist API `github-gist-api`
- VS Code Extension API `vscode-extension-api`

6.3 File and Class Dependency Graph Tools

- Doxygen with Graphviz `doxygen`, `graphviz`
- include-what-you-use (IWYU) `iwyu`
- LLVM Clang AST Visualization `clang-ast`
- CMake Graphviz Generator `cmake-graphviz`
- CodeViz `codeviz`