# Final Reflection
# Nodify: The Graph-based Python Visualizer

Team 7


| Full Name | Student Number | Email |
|---|---|---|
| Ahmet Erdem Dumlu | 400227350 | dumlua@mcmaster.ca |
| Anthony Hunt | 400297564 | hunta12@mcmaster.ca |
| Mutaz Helal | 400323509 | helala2@mcmaster.ca |
| Youcef Boumar | 400366531 | boumary@mcmaster.ca |


April 2, 2025

Link to Github: https://github.com/MutazHelal/nodify

# Achievements

At the beginning of the school year, with fresh motivation from our internships and real-world software engineering work, our group set out to create a tool to enhance the experience of developing software. We collectively recognized the difficulties of extending and maintaining large projects with minimal documentation and ever-changing requirements. Thus, the idea of Nodify was founded: a productivity-boosting tool to aid developers in navigating unfamiliar codebases. The three major components of Nodify are the backend static analysis, backend LLM integration, and frontend diagram webview within VSCode.

Throughout the first term, we formulated and implemented a plan to analyze and summarize source code files through purely automated methods. Although LLMs would inevitably play a sizable role in the feasibility of hands-free summary generation, we spent several weeks on static code analysis to procure concise, high-quality LLM inputs. Nodify's completed backend transforms plaintext Python/TypeScript code into a custom AST, organizes sections of code into hierarchical chunks, and maintains a complete record for the scope, location, and reference of every Python/TypeScript object.

Then, we integrated cloud LLMs to generate summaries for specific subsections of code chunks, balancing contextual knowledge of programming statements with input size for fast responses and practical use. Eventually, we realized that API tokens could get expensive quickly for large codebases, so we created a smaller fine-tuned LLM that could run on local GPUs with reasonable effectiveness. We found that small and medium sized codebases worked best with Nodify, but larger codebases would still produce acceptable diagrams for subsections of the full repository.

The last major feature of Nodify lies within VSCode integration for diagram output. Originally, we feared that diagrams for large codebases would be extremely messy and confusing for users, resulting in something entirely unhelpful for the development process. Thus, in Section 2.5.1 of the SRS, we outlined several levels of abstraction concerned with communicating different parts of the program. However, in practice, we found that the level 1 abstraction, wherein chunks of code are summarized and connected to related chunks, gave the most useful information without being overly messy. Directly translating the AST into diagram format, as specified by level 0, caused exponentially large diagrams, and abstracting further from code chunks reduced the overall usefulness of the diagram in its entirety. Consequently, we fully implemented only the level 1 abstraction layer and instead focused on providing clear pathing and navigation interactions for the graph.

# Requirements Completion

Almost all of the P0 requirements were completed or surpassed original specifications. Nodify successfully generates interactive flow diagrams for any Python (and even TypeScript) codebase with almost no incorrect information. Although the calibre of a generated diagram is

difficult to measure by normal LLM metrics, we have manually tested and fine-tuned Nodify's summarization capabilities to ensure semantic errors stay well below the 10% error threshold. Nodes are collapsable and draggable, with navigation through arrow keys (or even vim bindings). As described above, the original expectation of 4 different abstraction levels proved to be superfluous to the understanding of a codebase. Therefore, we have opted to set aside the abstraction level requirement in favour of optimizing the code-chunk-to-code-chunk diagrams.

Similarly, all requirements for P1 were satisfied, save for the aforementioned abstraction levels. We successfully published Nodify as a VSCode extension and integrated caching to reduce the number of LLM calls for similar sections of code. A smaller model was fine-tuned to enable high quality outputs on local consumer-grade GPUs, increasing the responsiveness of Nodify's graph generation and making Nodify accessible for developers without an OpenAI API key.

# Lessons Learned

Overall, our team is extremely happy with the progress and results of Nodify, even though the product is somewhat different from our original design. Through this capstone project, we gained a deeper understanding of programming language syntax and semantics, including symbol table construction for Python/TypeScript objects, optimal AST code chunking, and parsing input for a specific use case. We gained tangible, real-world experience working with LLMs, including memorable lessons on prompt engineering, the value of good input, and the direct translation of parameter count into output quality. We became familiar with React and VSCode's APIs, giving us a solid foundation for future UI-intensive projects. Least of all, we learned the process of balancing LLM optimization for speed and quality.

This project tested our technical and teamwork skills by necessitating component specialists and coming together to build a comprehensive system. Each aspect of Nodify is largely unique in the open-source software scene, requiring extensive research into adjacent programming language-, LLM-, and UI-specific tools. Separating the work among team members in this manner was effective but challenging to incorporate later on in the project. In fact, after all the main components were submitted, we had to almost-completely refactor and rewrite Nodify to add mirrored object tracking between the source code and diagram webview. Isolated development in this manner worked well for the first few months of this project, enabling us to focus on completing small sections of the overall problem, but the eventual combination of components required several days of code review to ensure everyone was on the same page.

Nodify has been an excellent experience in prototyping, iterating, and implementing a real-world project far beyond the scope of single-developer toy projects. We all enjoyed working on this project and appreciated the explorative, independent nature of this capstone course.