Architectural Enhancement: GNUstep

Queen's University
CISC 322
Instructor: Prof. Bram Adams
Winter 2025

Cameron Jenkins 20358084 21cj32@queensu.ca
Christine Ye 20354325 21cy27@queensu.ca
Evelyn Lee 20334769 21esl5@queensu.ca
Yiting Ma 20232544 20ym2@queensu.ca
Liam Beenken 20364179 22ltb1@queensu.ca

Table of contents

1. Abstract

In this paper we will present a comprehensive analysis of the added AI module for GNUstep. AI module is designed to add interfaces and abilities for developers to use AI-driven code assistant, which simplifies and speeds up the process of software development. This module also provides developers with powerful tools to integrate AI into the applications while maintaining compatibility with existing systems and other platforms. We will also describe the conceptual and architectural changes required to implement the AI module, the interactions with major GNUstep components (libs-base, libs-gui, libs-back, apps-gorm). A use case with sequence diagrams, various stakeholders like users, developers, etc. will also be provided. At the end, we will propose the implementation strategy and potential risks and mitigations for implementing this AI module.

2. Introduction

Since ChatGPT came out in 2022, AI has entered everyone's life. The software development industry has been greatly affected, and the gap between developers has narrowed dramatically. As an "old" framework, GNUstep should introduce some AI-related modules to facilitate developers who are still using this architecture to develop and use AI-related functions. At the same time, these modules also need to be able to connect with modern AI frameworks, such as Dify, Automa and other AI Agent and workflow frameworks. By adding AI abilities, developers can build smarter applications without switching frameworks. This update will help GNUstep stay relevant in this fast-changing world.

In our previous submissions, we have analyzed and explored the conceptual and concrete architecture of GNUstep. Based on the architecture we determined, and to enhance the functionality and developer experience, speed up the development process and enable the ability to adapt modern AI related functionalities to GNUstep, we have proposed a new library component called libs-ai. This module provides both the ability to accelerate software development and interfaces for developers to integrate artificial intelligent services into the application they are making.

We will provide detailed information for each feature we plan to integrate. Additionally, we will also demonstrate **two potential implementations** of this new module based on Layered style and Object-Oriented style. An updated conceptual architecture after we add the module will also be presented to show the new dependencies created by the implementation of this new module. At the same time, a SEI SAAM analysis will be provided to analyse the benefits and potential drawbacks of each approach to different major stakeholders and related NFRs. Then, we will choose one of the implementation plans and analyse the influences this module might have for other subsystems and components in GNUstep. Moreover, we will provide a use case that uses the newly added AI module with a sequence diagram to showcase the workflow. At last, we will summarize the content of this report and explain some lessons learned during the research and exploration of this module.

3. Proposed enhancement

The AI module of libs-ai integrates crucial AI-driven features into GNUstep, bridging a notable gap in the existing architecture and maintaining the framework's relevance in modern software development. This module covers Natural Language Processing (NLP) and Understanding, Computer Vision (CV) and Image Recognition, and Cloud-Based AI with LLM Integration. Using these characteristics, GNUstep programs may become more intelligent, automated, and developer-friendly while avoiding external dependencies and lengthy manual implementation.

GNUstep does not yet have built-in artificial intelligence capabilities, hence, developers must manually include other AI solutions, which are often ineffective and inconsistent. As the number of AI-driven applications in the software industry grows, frameworks that do not support AI become less competitive. Modern software frameworks increasingly include built-in AI capabilities, making them more attractive to developers that need intelligent automation, natural language processing, or picture recognition. Without these features, GNUstep programs remain limited in functionality, reducing their innovation potential. The libs-ai module directly solves this issue by offering a standardized AI solution that works cleanly with GNUstep's current architecture.

Natural language processing and interpretation are the foundation for all other AI capabilities. The AI module will provide pre-trained machine learning models, enabling developers to create NLP functionality using the libs-ai API. Text analysis, sentiment identification, speech-to-text conversion, machine translation, and artificial intelligence-assisted text prediction are available features. By adding NLP, GNUstep programs may enable better user interactions, improve accessibility using voice-based controls, and allow multilingual communication. Furthermore, NLP is required for AI-assisted code completion, which helps GNUstep developers write Objective-C more effectively by providing context-aware code recommendations and automatic mistake detection.

The CV and image recognition service includes interfaces for image recognition and computer vision applications such as optical character recognition (OCR), picture labelling, object and face identification, and gesture detection. Developers may include these functionalities using the libs-ai APIs, allowing programs to handle visual data more intelligently. This is particularly useful for document processing applications, security systems, and accessibility aids that require real-time picture analysis. These features enable GNUstep programs to automate procedures, extract information from photos, and improve user authentication via facial recognition.

Cloud-based AI capabilities and LLM integration are critical submodules of the AI module, allowing GNUstep applications to use current AI services without requiring significant local processing capacity. This integration provides a framework for developers to connect GNUstep programs to cloud-based LLM platforms, giving them access to generative AI models, workflow

automation tools such as Dify and Automa, and AI-powered assistants. One of the most important uses of LLM integration is co-pilot mode, in which the AI module helps developers write Objective-C by guessing user intent, making intelligent code recommendations, and creating sample code snippets. This increases developer productivity, decreases debugging time, and simplifies the software development process.

The libs-ai module is designed to integrate seamlessly into GNUstep's architecture, complementing existing components without interfering with core functionality. The AI module works with GNUstep's text input system for NLP-based recommendations, image processing library for CV tasks, and development environment for AI-assisted coding. By expanding on these core areas, libs-ai guarantees that GNUstep programs stay competitive and competent in tackling current AI-powered workloads.

4. Architectural Impact

Our key use cases for this additional module are AI integration support for developed GNUStep applications, and developer tools such as code completion and suggestions. A code completion tool may exist in many forms, and its implementation is often dependent on the IDE it is designed for. In development, we may consult reference examples such as the copilot extension for VS Code, however GNUStep includes its own specialized IDE within ProjectCenter. To promote the unique GNUStep related features it includes, we proposed an AI powered auto completion and suggestion plugin to be built for ProjectCenter. Additionally, in order to include support for developing AI powered applications on GNUStep, we proposed a new module, libs-ai, which will provide a useful API for including AI features in users' applications and some custom UI components built on top of those in libs-gui.

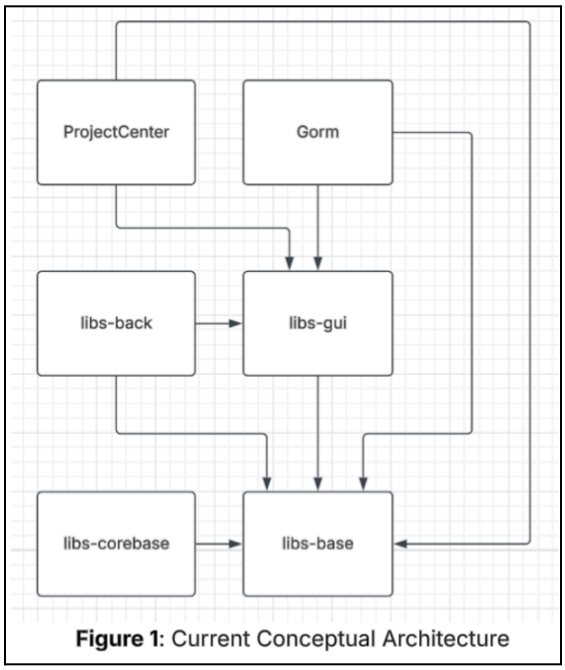

Figure 1: Current Conceptual Architecture

Figure 1 is our most recent conceptual architecture showing the relationships between relevant components, before adding our AI modules.

We have previously discussed the Layered and Object Oriented architecture styles used, ensuring that components are built on top of each other, encapsulate their own data, and provide a sole responsibility. These design fundamentals are important to the structure and maintainability of GNUStep, so we plan to keep them in mind when making design choices for the new module. Certain AI functionality is not feasible on users' hardware so calls to an externally hosted model will be necessary. Libs-base provides us with the core data structures and functions needed to make calls to a cloud API such as openAI's, so we will build libs-ai on top of libs-base. Additionally, to include the AI wrapped UI components, we

will include a dependency to libs-gui, and build on top of the already implemented elements. The final new dependency will be from ProjectCenter to libs-ai. This allows the code completion submodule to be built in ProjectCenter using the new AI functionality. Figure 2 shows the new proposed conceptual architecture, with the additional module and dependencies highlighted in red.

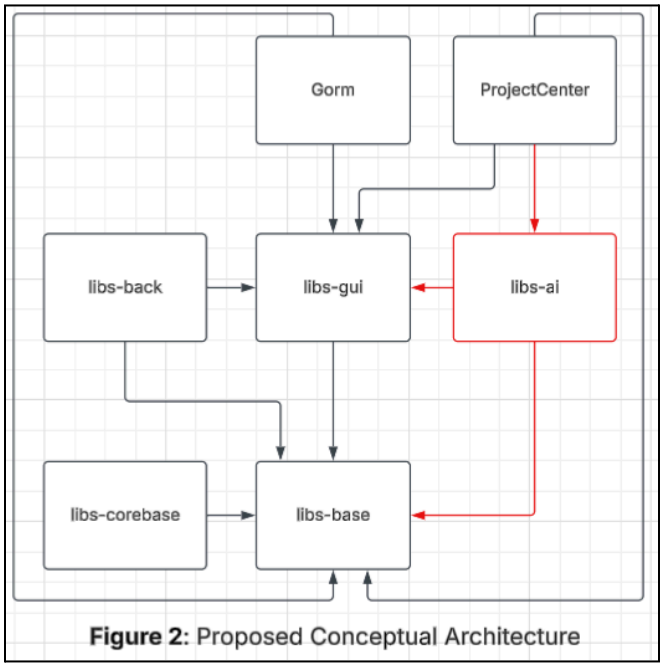

Figure 2: Proposed Conceptual Architecture

These changes maintain the layered style, as libs-ai does not rely on higher level components, and provides utility for the components above it. It also follows the object oriented principles of encapsulating all AI related utility in a single component, accessible to others via an exposed API. These additional dependencies will add optional utility to the development process, and should not add any unexpected performance overhead. The library will not be required in order to develop applications with GNUStep, only provide the support to easily include AI functionality, should it be desired.

On the low level side, this library will follow the design principles used in similar libraries, such as libs-gui. It will be designed to provide Objects and Methods with documented functionality, which can be imported into GNUStep projects. Some example Objects might include AIPredictiveTextField or AIChatbot. These Objects would be built on top of libs-gui elements, so any file that we build on top of would become a dependency. For example, libs-gui/Source/NSTextField.m would be imported into libs-ai/Source/AIPredictiveTextField.m. As mentioned previously, we would also build on top of the functionality in libs-base. This will flag many files as dependencies. For example, to make http requests to an LLM's cloud API, we we need to make use of libs-base/Source/NSURL.m[1], libs-base/Source/NSURLRequest.m[1], and libs-base/Source/NSURLResponse.m[1]. We will also need basic data structures and functionality to handle the data that we send and receive, such as NSArray[1], NSString[1], NSDictionary[1], and NSJSONSerialization[1]. All of these files would need to be included in relevant libs-ai files. Finally, we would need to include a new submodule in ProjectCenter for code completion and suggestions. This would require the code completion submodule to listen for the textDidChange() notification from the apps-projectcenter/Modules/Editors/ProjectCenter /PCEditor.m[3] file, retrieve the edited chunk of text using the current line number stored in the PCEditor Object, pass it to the codeCompletion() method in libs-ai, and display the result by adding the text to the buffer in a lighter shade. If the user likes the result, we add an extra KeyDown function to PCEditor.m[3] to handle accepting the suggestion.

5. Alternatives with SAAM analysis

In this section, we evaluate two different architectural approaches for integrating the new AI module into GNUstep. Each approach aims to add functionality for natural language processing, computer vision, and cloud-based LLM integration, but they differ in how tightly the AI capabilities are coupled with existing GNUstep components. We use the Software Engineering Institute's Software Architecture Analysis Method (SAAM) to perform the following steps:

Major Stakeholders

- **GNUstep Core Maintainers**: Responsible for maintaining the base libraries (libs-base, libs-gui, libs-back) and ensuring long-term viability of GNUstep.
- **Application Developers**: Build end-user applications on top of GNUstep, using tools such as Gorm and the newly proposed libs-ai library for NLP, CV, and AI-driven code assistance.
- **End Users**: Individuals or organizations running applications built on GNUstep that may leverage AI-driven features.
- **AI Framework Providers**: Offer cloud APIs (for example large language models, OCR services, image recognition) and integrate with the new AI module in GNUstep.

Key Non-Functional Requirements

Different stakeholders place emphasis on distinct NFRs:

1. **Maintainability**: GNUstep Core Maintainers want clean interfaces, minimal code duplication, and a manageable learning curve for new contributors.
2. **Performance**: Application Developers and End Users care about responsiveness of AI features, including speed of inference and low overhead when advanced AI features are not used.
3. **Security & Data Privacy**: All stakeholders require strong data protection, especially if cloud-based AI services handle sensitive information.
4. **Scalability & Extensibility**: As AI use grows, the solution must support increasing workloads or additional AI services without major architectural rewrites.
5. **Compatibility**: GNUstep has a long history. Core Maintainers and Application Developers want minimal breakage of existing applications and a straightforward upgrade path.

Evaluation of Two Alternative Implementations

Alternative A: Layered "Plugin-Style" Integration

In this approach, we treat the AI module (libs-ai) as a separate high-level layer above the core GNUstep libraries. The new library communicates with existing GUI and base libraries through well-defined APIs and plugin interfaces. Whenever an application requires AI functionality, it loads the relevant plugin or adapter at runtime.

- **Maintainability**: This alternative enforces strong separation of concerns, as libs-ai remains largely decoupled from core libraries. Maintainers can evolve libs-ai without heavily impacting libs-base or libs-gui. This structure also reduces the risk of regressions in other parts of GNUstep.

- **Performance**: There may be slight overhead due to plugin adapters and additional layers. However, if the plugin interfaces are efficiently designed and invoked only when AI features are explicitly required, overhead remains manageable.

- **Security & Data Privacy**: Centralized plugin loading can enforce encryption and data validation at a single entry point. A potential downside is that any misconfiguration in the plugin architecture could expose vulnerabilities if the plugin is not carefully sandboxed.

- **Scalability & Extensibility**: Because AI features are accessed via well-defined plugin points, new AI services (for example additional cloud providers) can be integrated without modifying the rest of the stack. This approach naturally scales as new models or AI tasks are introduced.

- **Compatibility**: Core GNUstep libraries see minimal changes, preserving backward compatibility. Existing applications remain functional and only need to adopt the plugin-based approach for new AI features if desired.

Alternative B: Object-Oriented Extension

In this approach, the project integrates AI features directly into existing GNUstep classes and modules. For instance, classes in libs-base or libs-gui might be subclassed or extended to include AI-aware methods (for example GSImage+AIRecognition, NSTextField+NLPCompletion). This creates a more seamless code experience for developers at the API level but increases coupling.

- **Maintainability**: Direct extension of existing classes increases coupling and complexity. Core maintainers must keep track of AI-specific logic interspersed within non-AI classes. This can complicate debugging and lead to regression issues if the AI codebase evolves rapidly.

- **Performance**: By integrating AI calls deeply into existing classes, performance could be optimized for certain tasks (for example local caching or specialized data structures). However, the risk is that even non-AI features become more complex, potentially slowing development and testing.

- **Security & Data Privacy**: If classes throughout the codebase handle AI requests, ensuring consistent data privacy and encryption may be more error-prone. Each extension point must implement robust security checks or data sanitization.

- **Scalability & Extensibility**: As additional AI services emerge, a direct extension approach requires scattering new methods in multiple classes. This can become unwieldy and hinder large-scale changes.

- **Compatibility**: Because the changes are more invasive, older applications may break if they rely on class hierarchies that are now extended with AI features. Wrappers or bridging code might be necessary to avoid conflicts.

Determination of Best Alternative

Based on the above analysis, Alternative A (Layered "Plugin-Style" Integration) seems to be the better choice. While direct OO extension can offer some performance benefits and potentially tighter integration, the plugin-style layer approach is easier to maintain over the long run, is more secure in a centralized manner, and allows new AI features to be added or upgraded without requiring invasive changes in the legacy parts of GNUstep. For most stakeholders, especially Core Maintainers and Application Developers, the lower architectural complexity and clearer boundaries outweigh the modest performance overhead. Thus, the layered plugin model is recommended for the new AI module in GNUstep.

6. Risk analysis

The proposed libs-ai module introduces powerful capabilities into GNUstep, enhancing its functionality with modern AI-driven features. However, it also brings a range of architectural and operational risks that must be carefully considered and managed to ensure long-term maintainability, performance, and system integrity. Addressing these risks is essential to preserve the stability and modular nature of GNUstep while supporting its evolution into a more intelligent and competitive development framework

Security risks

Integrating cloud-based AI services and large language models (LLMs) into GNUstep introduces several potential security concerns. These include the exposure of sensitive user data

when sent to external APIs, insecure communication channels if encryption is not properly enforced, and unauthorized access or misuse of AI services in the absence of robust authentication or rate-limiting. Additionally, injection vulnerabilities in NLP and image parsing models could lead to compromised behavior, and insufficient sandboxing of third-party model responses may pose further risks.

To address this issue, all data exchanges with cloud services must be securely encrypted using HTTPS/TLS. Access tokens should follow the principle of least privilege, and developers must implement input sanitization and AI-specific hardening, such as prompt filtering and output validation. Usage of secure, scope-limited APIs and sandboxed environments for processing model outputs will further reduce the attack surface.

Maintainability risks

The addition of the libs-ai module increases the architectural complexity of GNUstep and introduces several maintainability challenges. These include tighter coupling with existing components like libs-gui and Gorm, making the system harder to evolve independently. Debugging AI behavior may also become more difficult due to stochastic model outputs, while knowledge silos could form if only a few contributors understand the internal workings of the AI subsystem. Further, model versioning and lifecycle management pose risks—outdated or inconsistent models may degrade performance or lead to unexpected behavior. Additionally, changes in third-party cloud APIs used for LLM integration could break functionality if not properly managed.

To address these risks, the libs-ai codebase should be modularized with clean interfaces, allowing it to function independently of other subsystems. Comprehensive documentation, usage examples, and automated testing will help onboard contributors and ensure consistent behavior. Configuration files should manage model versions and endpoints, and automated regression tests should verify functionality across updates and integrations.

Performance risks

AI-driven features such as code prediction, image recognition, and LLM-based responses are computationally intensive and can introduce significant performance challenges within GNUstep applications. These include UI responsiveness issues caused by synchronous inference calls, increased memory and CPU usage from loading models or cloud payloads, and slow performance on legacy systems, which are common deployment targets for GNUstep.

To address these concerns, AI processing should be offloaded to background threads or handled asynchronously to avoid blocking the main UI thread. When available, GPU or NPU acceleration can improve execution speed. Additionally, lightweight fallback models and feature toggles should be implemented to adapt to a device's performance capabilities, ensuring broad compatibility and a smooth user experience across environments.

7. Testing and Evaluation Plans

To ensure the successful integration of libs-ai and its compatibility with GNUstep's architecture, we propose a comprehensive, multi-phase testing and evaluation strategy. This

approach aims to validate both the functional correctness of AI features and their alignment with key non-functional requirements (NFRs) such as performance, reliability, and usability.

Interaction testing

To evaluate the integration of libs-ai with the existing GNUstep architecture, we will conduct a series of tests focused on its interactions with core subsystems such as libs-gui, Gorm, and libs-base. These tests will begin with regression testing, ensuring that the introduction of the AI module does not interfere with or break any existing functionality within the framework. We will also perform mock integration tests, where libs-ai APIs are exercised through simulated user actions—for example, entering text to trigger NLP-based suggestions or uploading images for computer vision processing. These tests will be run within both Gorm and sample applications built on GNUstep to validate correct behavior in real-world contexts.

To assess system stability under load, we will carry out stress testing by issuing concurrent AI requests, such as multiple parallel code completion prompts or simultaneous inference operations. This will help us identify and resolve any performance bottlenecks or resource contention issues. Finally, we will test error handling and fallback mechanisms to ensure the system responds appropriately when AI services fail or become unreachable. This includes verifying that applications continue to function smoothly and degrade gracefully without crashing or producing unintended behavior.

Evaluation metrics

To measure the success of the libs-ai enhancement, we will use a combination of performance, accuracy, and usability metrics that align with both functional goals and non-functional requirements. Responsiveness will be measured by the time between user input and the AI-generated response, with a target latency of less than 200 to 250 milliseconds for tasks such as code completion and natural language interactions. Accuracy will be evaluated by comparing AI-generated outputs—such as code suggestions and image recognition results—against predefined benchmarks to determine the reliability of the module in real-world scenarios.

In addition, resource usage will be monitored by tracking CPU and memory consumption during both idle and active AI processing. We aim to limit performance overhead to no more than a 10 percent increase over baseline system usage. To assess maintainability, we will perform static code analysis and peer reviews focused on modularity, interface clarity, and documentation quality. Lastly, we will measure developer productivity by comparing the time required to complete common development tasks using AI-assisted features versus manual approaches. A noticeable improvement in task efficiency will indicate that libs-ai provides practical value in everyday development workflows.

Validation Against Non-Functional Requirements (NFRs)

To ensure that the libs-ai module meets stakeholder expectations, we have aligned our testing strategy with key non-functional requirements. For developers, maintainability is a primary concern. The module is designed with a plug-and-play architecture, allowing developers to easily integrate or disable AI features without modifying core application logic. For end-users, the focus is on performance and usability. AI-driven enhancements are implemented to be fast and non-intrusive, delivering intelligent responses with minimal latency to maintain a smooth user experience.

From the perspective of project maintainers, evolvability is critical. The architecture of libs-ai allows for easy upgrading or replacement of AI models without requiring a complete system overhaul. Additionally, the module maintains a clean separation of concerns, contributing to overall maintainability. Portability is also addressed by ensuring that AI features function consistently across all GNUstep-supported platforms. In cases where cloud services are unavailable, the system provides fallback mechanisms to maintain core functionality. These considerations help ensure that libs-ai supports long-term sustainability while delivering modern, intelligent features.

8. Use cases

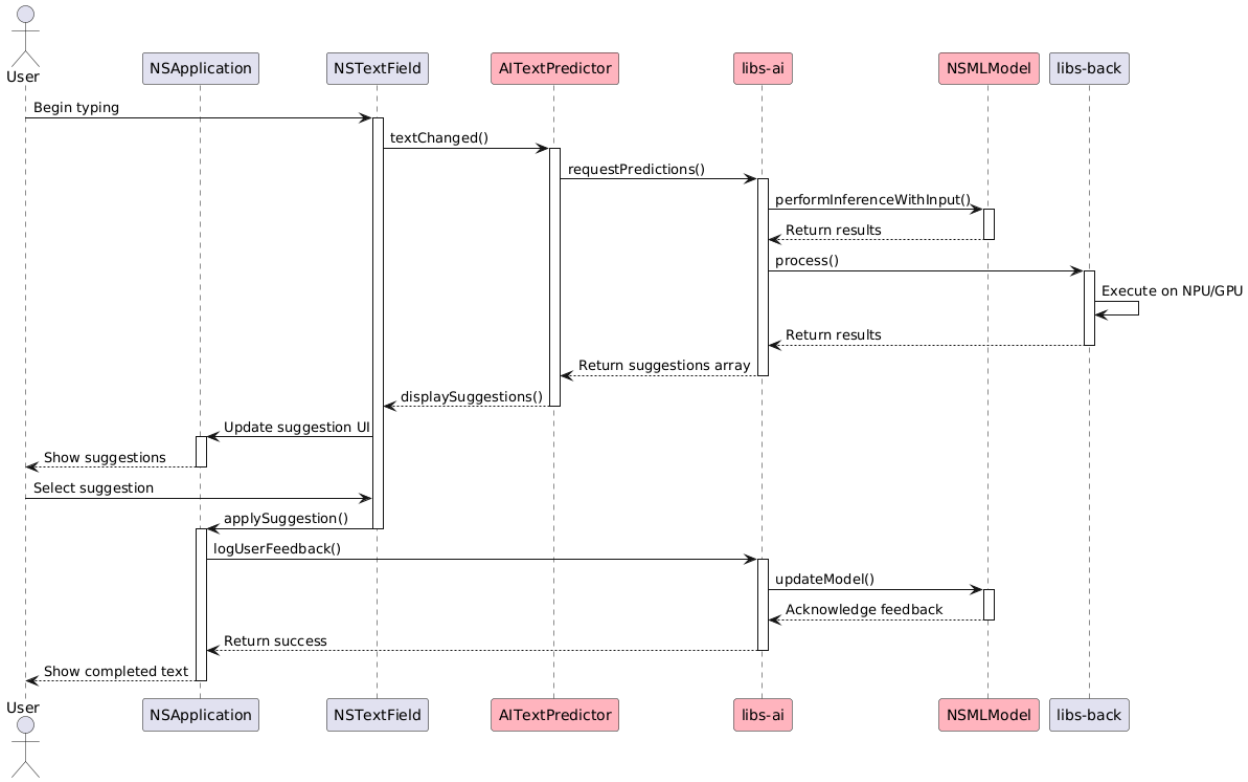

Figure 3. User Intention Prediction and Text Suggestion

Figure 3 shows an example use case of the user intention prediction and text suggestion when using an application made with GNUstep with AI module extensions.

At first, the user begins typing in a text field. We assume the text field has AI functionalities enabled. Text field will trigger a textChanged event and the AITextPredictor module will register and listen to this event. Once the event is triggered, it will use the prediction ability provided by libs-ai to run inference with user input. Libs-ai will use the results returned by language model and schedule NPU/GPU tasks to execute. After the results are returned from NPU/GPU, the results will be returned to libs-ai, and later on returned back to the user.Users can choose a suggestion from the results. The chosen result will be applied to the application, and the choice will be used to update the machine learning model.

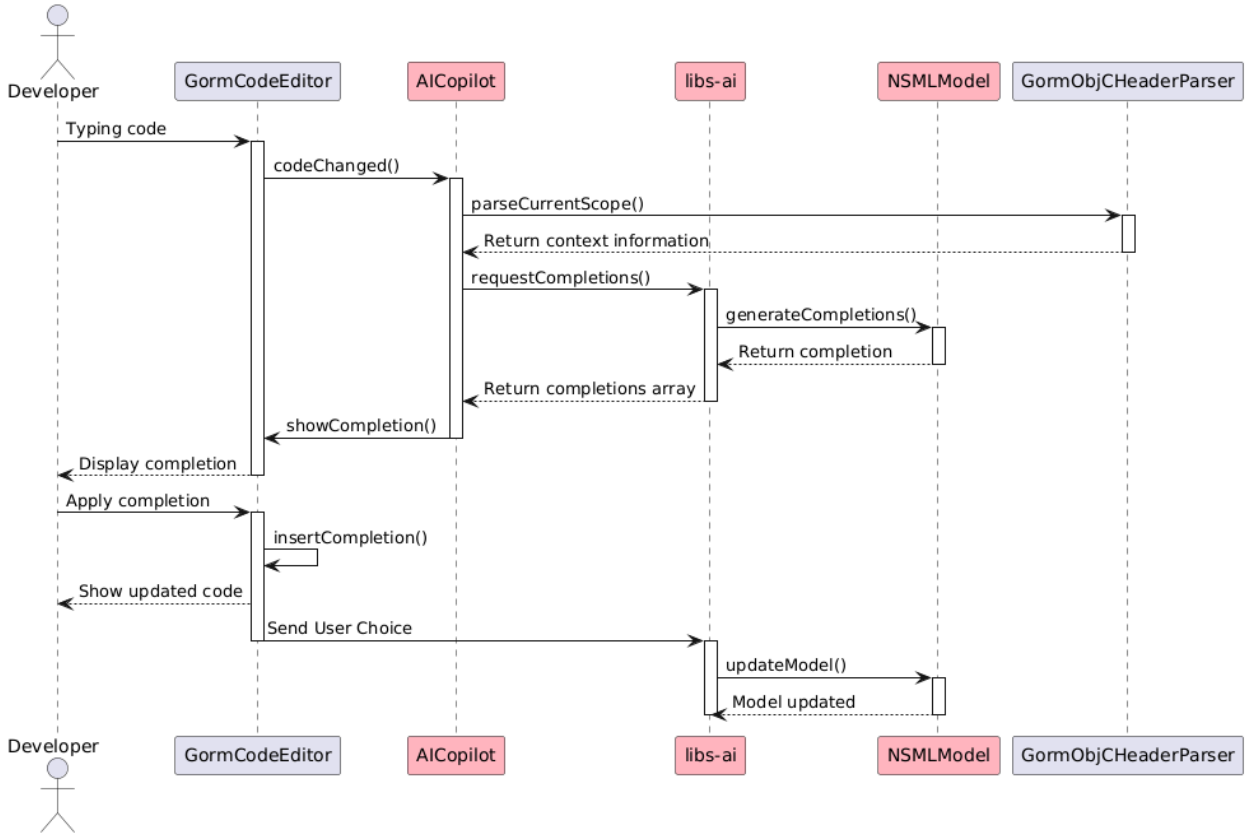

Figure 4. Copilot Code Auto-Completion

Figure 4 shows an example use case of the AI assisted code completion when developing applications using Gorm.

At first, the developer begins typing code in GormCodeEditor. We assume the code editor has AI code completion functionalities enabled. The editor will trigger a codeChanged event when the developer pauses typing, and the AICopilot module will register and listen to this event. Once the event is triggered, it will request the current code context from GormObjCHeaderParser to understand the scope and structure of the code. With this context information, AICopilot will use the code completion ability provided by libs-ai to generate relevant suggestions based on the current code and cursor position.

libs-ai will process this request by calling the language model to analyze code patterns and generate appropriate completions. After the results are generated, they will be returned to libs-ai, and then passed back to AICopilot. The assistant will then instruct the editor to display a completion popup with the suggestions. The developer can choose a suggestion from the results, and the choice will be updated to the AI model for better future suggestions.

9. Conclusion & Lesson learned

In conclusion, the new AI module added to GNUstep will provide abilities for developers to adapt modern AI-related functionalities to ongoing and existing applications made with the GNUstep framework. We have provided some use cases to show how this module benefits the product and development process, its potential impacts on the current framework and major stakeholders, and an analysis of the advantages and disadvantages this module might have to the entire system.

During the exploration and analysis process, we noticed that adding modern functionalities like the AI module is challenging because of the history of GNUstep. GNUstep is a relatively old framework that started around 1995. We have learned in lectures that Git came out in 2005, which means there are no git commits and logs for use to find between 1995 and 2005, which may include some important sticky notes and other information about subsystem dependencies in GNUstep. Secondly, the source code base of GNUstep is relatively large, which requires the team to spend a lot of time going through modules and subsystems to analyze the potential dependencies that the AI module might introduce to the entire system.

Finally, after the previous two submissions, our team has established a robust task distribution method that evenly distributes required tasks for deliverable A3. We have adapted the CPM method and project crashing method to deliverable A3 by separating tasks and defining the minimum and maximum time required for each task, then finding the critical path and starting working on the project. During the process of writing content for A3, we had some times when one task could not be done in time, so we applied the project crashing method by assigning another person who was working on a task that had some float to help on that task.

10. Naming Conventions

NLP: Natural Language Processing.

CV: Computer Vision.

OCR: Optical Character Recognition.

LLM: Large Language Model

ML: Machine Learning

GPU: Graphics Processing Unit

NPU: Neural Processing Unit

11. References

Gnustep. (n.d.-a). *Gnustep/libs-base*. GitHub. https://github.com/gnustep/libs-base

Gnustep. (n.d.-b). *Gnustep/libs-GUI*. GitHub. https://github.com/gnustep/libs-gui

Gnustep. (n.d.-b). *Gnustep/apps-projectcenter*. GitHub. https://github.com/gnustep/apps-projectcenter