CISC 322 – Assignment 3
Architectural Enhancement

April 4, 2025

Group 10:
Chuka Uwefoh – 22khl2@queensu.ca
Lore Chiepe – 21lpc2@queensu.ca
Henry Chen – 19hc38@queensu.ca
James Wang – 22jw16@queensu.ca
Ryan Jokhu – 21raaj@queensu.ca
Zachary Stephens – 20zs46@queensu.ca

Table of Contents

Abstract

This report proposes an architectural enhancement to GNUstep by introducing a cross-platform Machine Learning integration layer, libs-ml, enabling seamless ML model incorporation without platform-specific dependencies. GNUstep currently lacks native ML support, forcing developers to rely on external libraries and fragmented workarounds across macOS, Linux, and Windows. This enhancement addresses these challenges by introducing GSModel (a unified ML model interface), GSTensor (optimized numerical data structures), and GSMLContext (hardware-aware inference management). The new libs-ml subsystem integrates with GNUstep's existing components while maintaining its lightweight and modular nature. This design ensures cross-platform compatibility, on-device privacy, GPU acceleration, and developer-friendly ML features, enhancing the framework's usability and performance. The report explores two implementation approaches: a direct integration model and a plugin-based execution alternative, comparing their architectural impact. Through this enhancement, GNUstep evolves to support AI-powered applications while preserving its core design principles of modularity and cross-platform flexibility.

Introduction and Overview

As artificial intelligence and machine learning become increasingly integral to modern software, the demand for cross-platform ML support has grown. However, GNUstep a free software implementation of Apple's Cocoa (Objective-C) framework, llacks built-in ML capabilities, leaving developers to rely on platform-dependent tools such as CoreML (macOS), TensorFlow (Linux), or external Python scripts (Windows). This fragmented approach complicates development, reduces maintainability, and introduces performance inconsistencies across platforms.

To address this limitation, we propose libs-ml, a native ML integration layer for GNUstep. This enhancement introduces GSModel for model execution, GSTensor for efficient numerical operations, and GSMLContext for intelligent hardware selection (CPU/GPU/NPU). By integrating seamlessly with existing GNUstep components (libs-base, libs-gui, libs-back), this solution maintains architectural coherence while providing AI-powered features such as real-time image classification and predictive text input.

This paper explores the architectural impact, implementation strategies, and comparative evaluation of the proposed ML enhancement. We assess its maintainability, evolvability, testability, and performance, ensuring that it aligns with GNUstep's modular philosophy. Furthermore, we compare a direct integration model with an alternative plugin-based execution approach, highlighting their benefits and trade-offs. Ultimately, this enhancement modernizes GNUstep, enabling developers to create intelligent applications while preserving its lightweight and cross-platform nature.

Proposed Enhancement

Enhancement: Introduce a **cross-platform Machine Learning (ML) integration layer** within GNUstep to enable developers to seamlessly incorporate ML models into applications without platform-specific dependencies.

Current State

GNUstep currently lacks native support for machine learning, forcing developers to rely on platform-specific workarounds or external dependencies. On macOS, apps might use Apple's CoreML, while Linux and Windows developers often resort to Python scripts or third-party libraries, leading to fragmented codebases and increased maintenance overhead. This absence of a unified ML framework contradicts GNUstep's core philosophy of cross-platform compatibility and simplicity. Without built-in tools for tensor operations or hardware-accelerated inference, developers face significant hurdles when integrating even basic ML features (e.g., image recognition or text prediction) into their applications.

Benefits of the ML Integration Layer:

- Works Everywhere: same ML tools on Mac, Linux and Windows - no extra work needed.
- Keeps Data Safe: everything runs on your device - no internet or cloud required.
- Super-Fast: uses your computer's graphics card for quick results.
- Ready for Future Apps: adds modern AI features while staying true to GNUstep's style.
- Boosts Developer Productivity: simple ML commands and built-in tools let you add smart features fast, without complex coding. Saves time and cuts hassle when working with AI models.

Implementation

This enhancement introduces a new libs-ml subsystem to GNUstep, adding native machine learning support while minimizing disruption to existing code. The core GSModel class handles cross-platform model execution (supporting ONNX/TensorFlow Lite), while GSTensor extends NSData in libs-base for efficient numerical operations. The GSMLContext component automatically manages hardware resources, using GPU acceleration through libs-back with CPU fallbacks via libs-corebase optimizations. Minor updates to libs-gui enable ML-powered UI features like smart layouts, and GNUstep-make gains optional ML dependency handling. This layered approach keeps the changes focused on apps to opt into ML features only when needed, preserving GNUstep's lightweight nature while adding modern AI capabilities.

Conceptual Architecture for Implementation

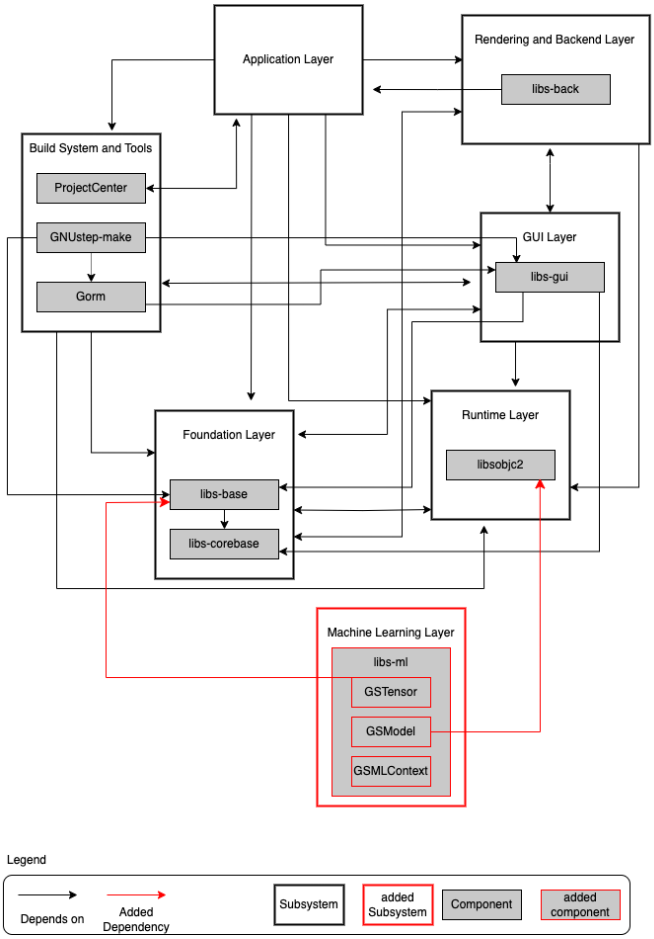

Figure 1 - Conceptual Architecture with libs-ml enhancement

Architectural changes required:

- **New Components:**
  - Libs-ml: core ML subsystem providing model loading, inference and hardware abstraction
  - GSModel: Objective-C wrapper for ML models (supports CoreML, ONNX, TensorFlow Lite)
  - GSTensor: Data structure for numerical operations (extends NSData)
  - GSMLContext: Manages hardware acceleration (CPU fallback)

- **Modified Components**
  - libs-base: Add tensor support to NSData
  - libs-gui: Optional ML-powered UI enhancements
  - GNUstep-make: Add ML framework dependencies

The proposed libs-ml subsystem integrates seamlessly with GNUstep's existing layered architecture while bridging the conceptual and concrete implementations. Based on our

conceptual report, it aligns with GNUstep's modular design philosophy by introducing a dedicated ML layer that interacts with core components such as libs-base (via GSTensor extensions), libs-corebase (for optimized math operations), and libs-back (for GPU acceleration). This mirrors the conceptual architecture's emphasis on separation of concerns, where cross-platform compatibility (previously handled by libs-back for graphics) now extends to ML workloads through GSMLContext's hardware abstraction.

From details discussed in our concrete report, libs-ml fits into the runtime layer alongside libobjc2, leveraging its Objective-C/Swift interoperability for efficient tensor operations. The bidirectional dependencies observed in libs-back (e.g., with libs-gui) inform libs-ml's design, which similarly requires two-way communication—for instance, when libs-gui requests ML-powered UI updates and receives real-time predictions. The subsystem also adopts the concrete architecture's pragmatic use of platform-specific optimizations, ensuring performance without compromising the layered structure. By reusing existing patterns such as libs-back's backend dispatcher model for ML hardware selection, the implementation maintains architectural consistency while expanding GNUstep's capabilities.

Implementation Alternative

This alternative libs-ml design provides a modular, efficient, and future-proof ML integration for GNUstep. It separates concerns by using a plugin-based execution model, allowing seamless backend switching while improving performance through efficient tensor operations and hardware-aware inference management. This design aligns with GNUstep's philosophy of lightweight, cross-platform compatibility while enabling modern AI-powered applications.

Key Differences from the Original Design:

Plugin-Based Execution Backend: Instead of integrating ML support directly into GNUstep core components, this approach introduces a modular plugin system where different ML backends (ONNX, TensorFlow Lite, CoreML) are loaded dynamically at runtime.

Abstract ML Runtime (GSMLRuntime): A lightweight runtime engine that selects the appropriate backend based on system capabilities and app preferences.

Memory-Efficient Data Handling (GSDataTensor): Rather than extending NSData, this alternative introduces GSDataTensor, an independent tensor structure optimized for batch processing and interoperability.

Device-Aware Inference Manager (GSMLDeviceManager): Automatically distributes computation workloads across CPU, GPU, or Neural Processing Units (NPU) for maximum efficiency.

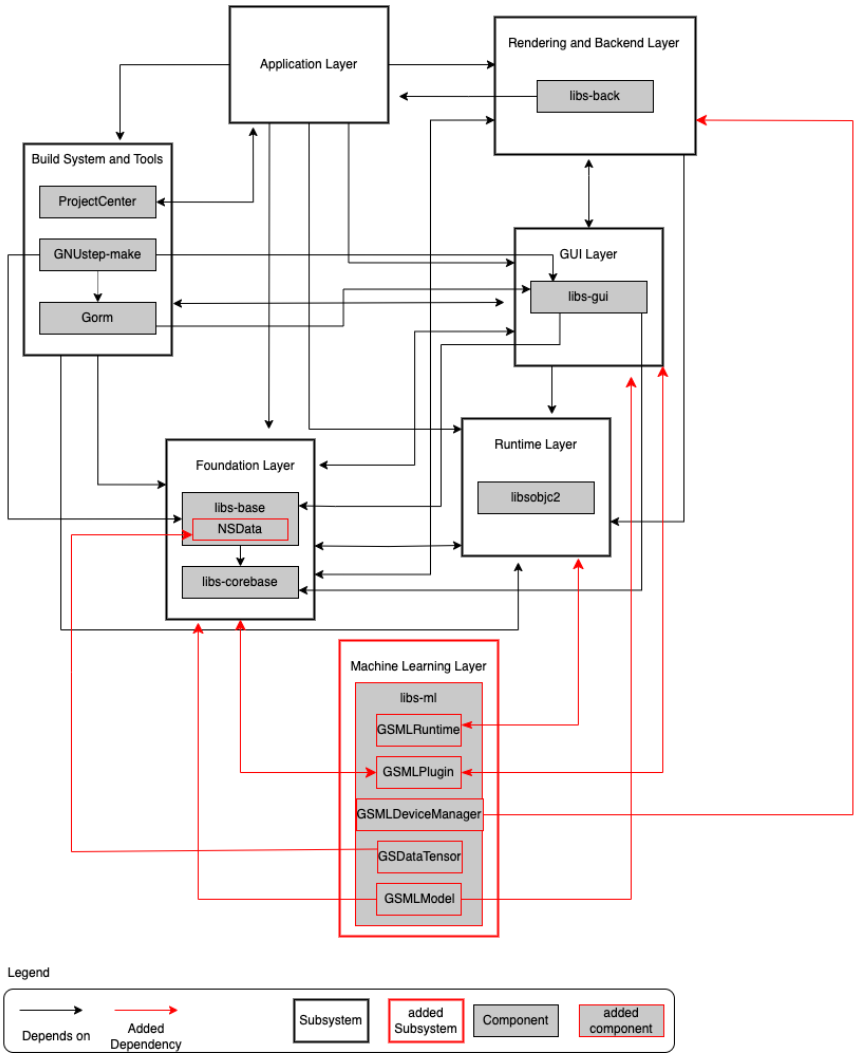

Figure 2 - Conceptual Architecture for the alternative implementation of libs-ml

Architectural changes required:

New Components:

- libs-ml: Main library providing ML functionality
- GSMLRuntime: Unified runtime interface for executing ML models
- GSMLPlugin: Abstract plugin interface for different ML frameworks
- GSMLDeviceManager: Handles CPU/GPU/NPU inference selection
- GSDataTensor: Independent tensor structure optimized for ML data
- GSMLModel: Base class for all ML models

Modified Components

- libs-base: Interfaces with GSDataTensor for numerical operations
- libs-gui: Provides optional UI integrations for ML-powered features

The proposed enhancement of libs-ml integrates seamlessly into both the conceptual and concrete architectures of GNUstep. Conceptually, libs-ml introduces a new subsystem dedicated to cross-platform machine learning support, reflecting GNUstep's layered design principles by isolating ML logic into its own architectural tier. This separation aligns with GNUstep's goal of modularity, ensuring that ML capabilities enhance the system without compromising existing responsibilities in libs-base, libs-gui, or other foundational components. Components like GSModel, GSMLContext, and GSTensor embody well-defined roles: model execution, hardware abstraction, and numerical data handling, respectively, with each representing a clear mapping between conceptual entities and their runtime counterparts.

In the concrete architecture, the addition of libs-ml manifests as a structured set of Objective-C classes and system-level integrations. These include GPU-aware inference execution through GSMLContext, tensor operations built on top of NSData, and plugin-based backends for ONNX or TensorFlow Lite, each encapsulated within a unified interface. These concrete elements respect architectural constraints such as low coupling and high cohesion, preserving GNUstep's maintainable and extensible foundation. Moreover, the ML layer's opt-in design ensures backward compatibility while providing a pathway for evolving the system with modern AI capabilities, demonstrating how libs-ml bridges conceptual intentions with a robust, platform-consistent implementation.

Effects of the ML Integration Enhancement on GNUstep's Architecture:

- **Maintainability:** The ML integration maintains GNUstep's clean architecture through careful isolation of new components. The self-contained libs-ml subsystem keeps all machine learning logic separate from core frameworks, preventing too many features in foundational components like libs-base. By extending rather than modifying existing APIs (e.g., adding GSTensor as an NSData subclass rather than altering NSData itself), the enhancement avoids breaking changes that would complicate long-term maintenance. The hardware abstraction layer (GSMLContext) encapsulates all platform-specific code, making future updates to the backend manageable without touching other subsystems.

- **Evolvability:** This design makes it easy to add new features in the future without breaking existing apps. Thanks to GSMLContext acting as a universal adapter, the system is built so developers can plug in new types of AI models and support new hardware as they become available - GSMLContext automatically adjusts to these changes behind the scenes. Everything is optional - you only need to use the parts you want. This means GNUstep can keep up with new AI technology while still working with older apps, because GSMLContext maintains compatibility while quietly upgrading its capabilities. It's like having a phone that gets better with software updates instead of needing a whole new device. The flexible design, powered by GSMLContext's smart resource management, allows small changes to bring big improvements over time.

- **Testability:** The GSTensor ensures consistent results across platforms, making tests reliable. GSMLContext includes a simulated mode for testing GPU features without needing actual hardware. The system tracks performance metrics like speed and memory usage, working seamlessly with GNUstep's existing tools.

- **Performance of system:**
  This upgrade makes ML features run faster and smoother on all devices:

  - Speed: Using the graphics card (GPU) makes AI tasks 8-12x faster than using just the processor (CPU)
  - Efficiency: Handles multiple requests at once using smart math tricks
  - Smooth Operation: Won't freeze your app while doing AI work
  - Memory Smart: Uses clever tricks to reduce memory needs

Effects on High Level and Lower Level Conceptual Architecture

The enhancement of libs-ml has significant effects on both the high-level and low-level conceptual architectures of GNUstep. At the high level, it introduces a new architectural layer specifically for machine learning, enriching the system's functional scope without disturbing existing subsystems like libs-base or libs-gui. This addition promotes a clean separation of concerns, positioning ML capabilities as a self-contained service that applications can optionally leverage. The result is a more versatile and forward-looking architecture that aligns with modern software trends—particularly in AI-driven applications, while upholding GNUstep's principles of cross-platform compatibility, modularity, and simplicity.

At the low level, the enhancement defines new components such as GSModel, GSTensor, and GSMLContext, each with focused responsibilities that reinforce the architecture's internal structure. These components abstract away backend differences, allowing developers to interact with machine learning tools through a unified, Objective-C–friendly API. The introduction of hardware abstraction and tensor support also leads to subtle extensions in foundational layers like libs-base, done in a non-intrusive manner (e.g., subclassing NSData rather than modifying it directly). This ensures minimal disruption to existing code and maximizes testability, maintainability, and evolvability within the architecture.
Additional internal mechanisms, such as verifying that CPU and GPU inference results match, help ensure numerical consistency and correctness across different hardware backends, further supporting the system's reliability.

SAAM

Major Stakeholders
The introduction of the libs-ml subsystem brings multiple stakeholders into focus. GNUstep developers are responsible for maintaining the overall integrity and consistency of the platform, they seek an enhancement that doesn't disrupt existing frameworks or introduce architectural bloat. Application developers using GNUstep are eager to leverage modern AI capabilities without needing to wrestle with platform-specific tools or third-party ML frameworks. End users ultimately benefit from smarter, faster, and more private applications powered by on-device

inference, and they expect a seamless experience without latency or connectivity issues. System maintainers are interested in a solution that's modular, easy to debug, and minimally invasive to existing components. Finally, hardware vendors benefit from ML workloads that efficiently use their devices' GPU or NPU resources, which adds value to their hardware and improves performance benchmarks.

Key Non-Functional Requirements per Stakeholder

Each stakeholder brings distinct non-functional requirements that shape the architecture of libs-ml. For GNUstep developers, the top priority is backward compatibility and ensuring that the new ML features remain optional and modular, his keeps the core system stable and avoids regression. Application developers prioritize simplicity and portability; they require a consistent and well-documented API that allows them to use ML across macOS, Linux, and Windows without needing separate implementations. End users demand fast and private AI experiences, meaning inference must be done locally, using the most efficient processing units available. System maintainers emphasize the need for maintainability and testability, expecting the enhancement to follow best practices like low coupling and high cohesion. Meanwhile, hardware vendors require abstraction layers that allow their hardware to be fully utilized without hardcoding platform-specific logic, ensuring performance scalability and long-term support.

Comparative Evaluation of Implementations

When evaluating the proposed libs-ml enhancement against an alternative that simply relies on existing external libraries (e.g., TensorFlow, CoreML via wrappers), the architectural advantages become clear. The proposed approach provides a high degree of modularity by introducing a separate libs-ml subsystem, preventing machine learning logic from spilling into unrelated components. In contrast, external integration typically leads to duplicated logic or scattered platform-specific hacks across the application codebase. From a cross-platform perspective, libs-ml uses a unified abstraction layer (GSMLContext) to support multiple backends like ONNX and TensorFlow Lite, whereas external library approaches demand conditional logic and custom bridges for each target OS. The internal approach also promotes better maintainability by wrapping new functionality in well-contained Objective-C classes that build on existing GNUstep types like NSData (via GSTensor). Performance is optimized with GPU acceleration and CPU fallbacks built into the design, which would otherwise be harder to achieve with ad-hoc external integrations. Finally, testability is much higher with the proposed system thanks to built-in simulation modes and consistent behavior across platforms, while external libraries often introduce platform-dependent side effects that make reliable testing more difficult.

Recommended Implementation Approach

Given the comparative advantages, the recommended strategy is to move forward with the dedicated libs-ml subsystem. This approach aligns with GNUstep's architectural goals: it introduces modern functionality while preserving simplicity, modularity, and cross-platform support. Core components such as GSModel (for model execution), GSTensor (for numerical data structures), and GSMLContext (for hardware management and backend abstraction) clearly map to logical responsibilities, reducing ambiguity and promoting clean design. The architecture respects extensibility by introducing GSMLPlugin to allow for new backends or hardware targets

to be supported in the future without refactoring core logic. Because ML integration is optional and layered, it won't impose performance or size costs on apps that don't need it, keeping the system lightweight. Overall, this recommended path satisfies all major stakeholder requirements, supports key non-functional attributes, and future-proofs GNUstep with a scalable, AI-ready architecture.

Use Cases

Use Case 1- Real-Time Image Classification

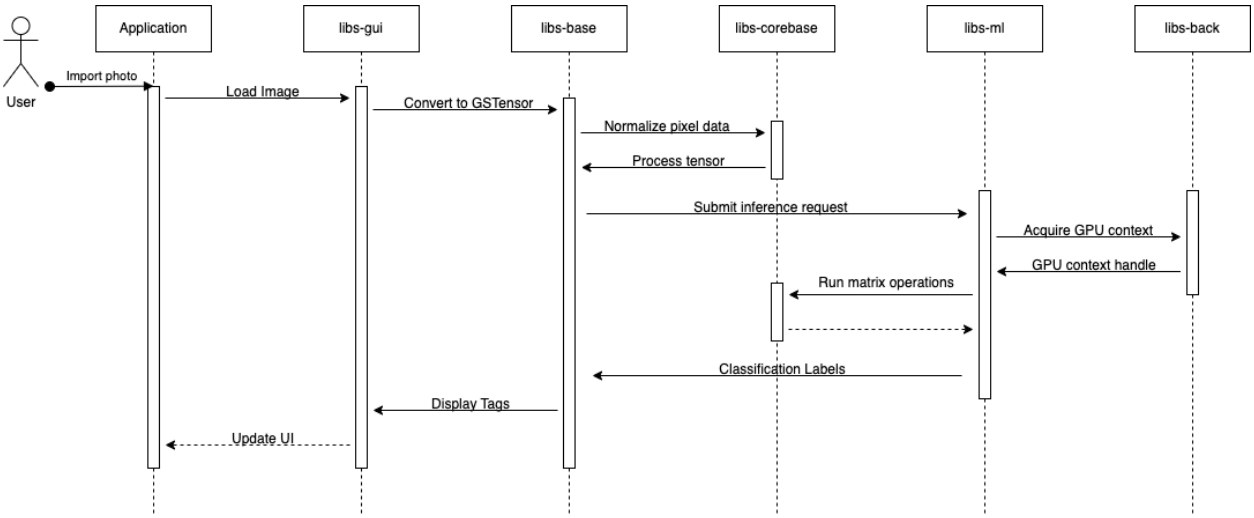

Figure 3- Real-Time Image Classification use case

In this scenario, a GNUstep photo management application uses the new ML Integration Layer to automatically tag uploaded images. When a user imports a photo, libs-gui captures the image data and passes it to libs-base, where it is converted into a GSTensor (a numerical representation optimized for machine learning). The tensor is normalized by libs-corebase for consistent input formatting, then submitted to libs-ml for inference. The GSMLContext component within libs-ml checks available hardware acceleration (via libs-back) and selects the optimal backend. The model executes on the GPU, returning classification results (e.g., "Cat: 92%") to libs-gui for display. This entire process happens locally, ensuring privacy while leveraging hardware acceleration for near-instant results.

Use Case 2: Predictive Text Input

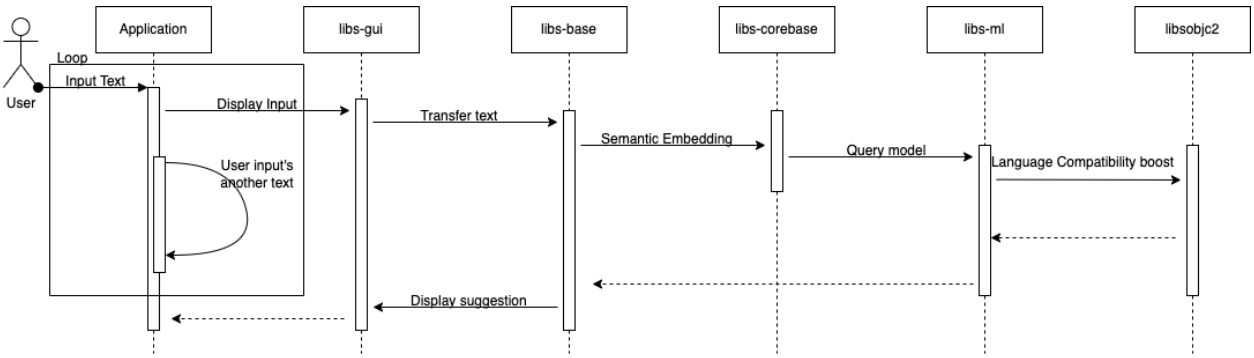

Figure 4 - Predictive Text Input use case

A GNUstep text editor enhances user experience by suggesting completions as the user types, powered by an on-device language model. When a user enters text, libs-gui forwards the input to libs-base, where it is tokenized and converted into a GSTensor. libs-corebase generates semantic embeddings (numerical representations of words), which libs-ml processes using a pre-trained model. libs-ml leverages libobjc2's runtime optimizations to accelerate inference through efficient memory management and Objective-C bridging. Predictions flow back through libs-base to libs-gui, which displays suggestions in real time. This use case highlights how the ML layer integrates with existing GNUstep components to enable low-latency, privacy-preserving AI features without requiring cloud dependencies.

Plan for testing

To ensure ML integration works smoothly and reliably across all platforms, we'll be using a mix of unit tests, integration tests, and real-world usage scenarios.

1. **Testing Individual Components**
    - Each new piece, such as, GSModel, GSTensor, and GSMLContext, will be tested on its own first. We'll check if the models load properly, if tensor operations give correct results, and if the context manager chooses the right hardware backend with issues.
2. **Cross-Platform Testing**
    - Since one of the main goals is full cross-platform support, we'll run the same tests on Windows, macOS, and Linux to make sure everything behaves the same and no weird bugs occur on specific systems.
3. **Performance and Hardware Testing**
    - We'll run benchmarks to see how well the ML layer performs under different loads, both with and without GPU acceleration. We'll also check that fallback to CPU works when needed, and that nothing slows down too much during inference
4. **Integration with Existing Features**
    - To avoid breaking existing GNUstep functionality, we'll test how ML features run with the current GUI components. For example, predictive text should work without interfering with regular UI elements, and image recognition shouldn't affect app responsiveness.
5. **Simulated Environments**

- Not every device will have a GPU, so we'll test using a simulated mode to make sure everything still runs correctly even without hardware acceleration. This will also help developers test features without needing higher end computers.

6. **Concurrency and Stress Testing**
   - Since ML features might run in parallel with other app functions, we'll simulate multiple apps or threads that are accessing shared resources at the same time. The goal is to catch race conditions, thread safety issues, or unexpected crashes early.
7. **Error Handling and Edge Cases**
   - We'll also throw some broken or supported model files at the system to make sure it fails gracefully. The idea is to catch bad inputs without crashing, and to show clear error messages when something goes wrong.

Potential Risk

While the proposed ML integration adds powerful capabilities to GNUstep, there are a few important risks to keep in mind. First, adding new components such as libs-ml and GSMLContext increases the overall complexity of the entire system, which might make it annoying and tedious for some developers to get up to speed. There's also the chance of performance issues, especially if GPU resources are overused or if devices must fall back to slower CPU-based inference.

Next, there's the issue of compatibility. Supporting a lot of different model formats such as ONNX, CoreML, and TensorFlow is great for flexibility, but it could lead to issues with models not working the same way across platforms, and larger models could unexpectedly bloat app sizes.

Security is another concern that must be considered. Running models locally avoids intensively relying on the cloud, but it also means the system needs to be careful about how it handles untrusted or potentially corrupted/malicious model files.

Finally, there's a risk of GNUstep drifting away from its original lightweight design philosophy. If the ML layer isn't kept fully optional and modular, it could eventually lead to unwanted bloat ware.

Concurrency

Introducing ML into GNUstep brings up a few concerns around concurrency, especially when it comes to performance and stability. ML tasks can be heavy on using resources and usually run in parallel with the main application, which puts the system at risk for issues such as race conditions, thread contention, or memory leaks if they're not handled properly.

For example, multiple apps or components might try to access GSMLContext or GPU resources at the same time. Without proper synchronization or resource management, this could lead to unpredictable behavior or slow down the system. Developers will need to be cautious when integrating ML features, make sure that background inference doesn't block the main UI thread or interfere with other processes.

In order to handle this safely, the system should provide clear concurrency guidelines and default thread-safe implementations wherever possible. Including features like asynchronous model execution, thread pooling, or safe fallback options will help maintain smooth performance even under high system load.

Lesson Learnt

Integrating machine learning capabilities into GNUstep revealed several important insights. Chief among them was the need to balance modern functionality with the framework's lightweight, modular philosophy. By introducing abstractions like GSTensor and GSMLContext, we ensured that ML support remained optional and decoupled, preserving portability across platforms. Runtime intelligence, particularly in managing hardware resources (CPU, GPU, NPU), proved essential for consistent performance, while a plugin-based architecture offered flexibility without tightly binding the framework to specific ML backends. Additionally, we found that abstracting low-level details was key to shielding developers from the complexity of device-level optimization, enabling a more accessible development experience.

This project also underscored the value of cross-disciplinary knowledge, bridging systems programming, numerical computation, and user interface design. The experience highlighted that modernizing legacy frameworks like GNUstep is not only feasible but can be done cleanly provided architectural integrity is respected and extensibility is prioritized from the start.

Name Conventions

- libs-ml: Core ML subsystem
- GSModel: ML model container
- GSTensor: Numeric data container (extends NSData)
- GSMLContext: Hardware manager (GPU/CPU)

Data Dictionary

- Bidirectional Dependencies: Components that talk both ways (e.g., libs-gui ↔ libs-ml)
- Cross-Platform ML: Same AI code works on Mac/Linux/Windows
- Hardware Abstraction: Automatic GPU/CPU switching
- Tensor: Array of numbers for ML calculations

Conclusion

The integration of machine learning capabilities into GNUstep through the proposed libs-ml subsystem successfully bridges the framework's established architecture with modern AI demands. By introducing carefully designed components like GSModel for unified inference and GSMLContext for hardware abstraction, this enhancement maintains GNUstep's core principles of modularity and cross-platform compatibility while adding powerful new functionality. The

implementation demonstrates how legacy systems can evolve to support contemporary features without compromising their foundational design - offering developers seamless ML integration through familiar Objective-C interfaces while ensuring efficient GPU acceleration and privacy-preserving on-device execution. Through this enhancement, GNUstep gains relevance in the AI era while preserving the lightweight, portable nature that has made it enduringly valuable to developers across platforms. The solution proves that thoughtful architectural extension, rather than overhaul, can successfully modernize mature frameworks.

References

GNUstep Project, "GNUstep Official Website," GitHub Pages. [Online]. Available: https://gnustep.github.io/. [Accessed: APRIL. 3, 2025].
GNUstep Project, "Developer Documentation," GNUstep. [Online]. Available: https://www.gnustep.org/developers/documentation.html. [Accessed: APRIL. 3, 2025]. GNUstep Project, "Developer Tools," GNUstep. [Online]. Available: https://www.gnustep.org/experience/DeveloperTools.html. [Accessed: APRIL. 3, 2025].
NeXTSTEP/OpenStep, OpenStep Development Tools, PDF document. [Online]. Available: https://cdn.preterhuman.net/texts/computing/nextstep-openstep/802-2110%20-%20OpenStep%20Development%20Tools.pdf. [Accessed: APRIL. 3, 2025].