Program Increment (PI) Plan: LMForge

Project: LMForge - Locally Hosted LLM Builder **PI Duration:** Fall Semester 2025 (5 Sprints)

1. Executive Summary & Project Evolution

Vision: To democratize AI development by providing a seamless, privacy-conscious platform ("LMForge") that enables non-technical users to scrape data, clean it, and fine-tune open-source models for domain-specific tasks without relying on cloud infrastructure.

Project State: Current Status (Backend & Frontend Focus)

Feature Area	Current State (End of PI)
Backend Architecture	**Microservices Architecture:** Five distinct backend modules (FastAPI, Django, Flask) that process data independently.
User Interface (UI)	**Unified Design System:** A specialized cross-functional sub-team integrated a polished, consistent React-based UI across all modules based on a validated Figma prototype.
Data Pipeline	**Functional Silos:** The system can successfully Scrape (Reddit/Web), Clean (Pandas), and Vectorize (pgvector), though the handoff currently requires manual file transfer.
Infrastructure	**Hybrid Containerization:** RAG and Database are fully Dockerized; Agentic uses MCP; Fine Tuning & Scraping run in local Python environments due to resource constraints.
Key Metrics	**36% Reduction** in token usage (Cleaning Team); **100%** Success rate on Agent tool calls; **0%** Duplicate headings in processed data.

2. Team Roster & Technical Achievements

A deep dive into the specific backend logic, architectural choices, and deliverables defined in the final team documentation.

Team	Key Deliverables & Architecture
UI/UX Design Sub-team *(Cross-functional Tiger Team)*	• **Deliverable:** Created a comprehensive **Figma Prototype** to address the initial lack of visual polish. • **Integration:** Successfully translated the Figma design into a unified **React Frontend**, replacing disjointed team-specific interfaces.
Agile Agentic Systems	• **Solution:** "LMForge Agent Creator" — A no-code tool to build agents. • **Architecture:** Implemented **FastAPI** to serve agent logic and adopted **MCP (Model Context Protocol)** to standardize tool calling. • **Methodology:** Used **BPMN** to align team goals and resolve initial scope ambiguity. • **Tech Stack:** Python, LangChain, MCP-Use, Figma.
Web Scraping	• **Ingestion Engine:** Built a multi-threaded Python scraper using **Selenium** and **Reddit API**. • **Compliance:** Focused on "Legal Scraping" by prioritizing official APIs over brute-force methods. • **Logic:** Implemented batch processing for PDFs and URLs.
Data Cleaning	• **Algorithms:** Developed "Content-Aware" cleaning scripts that utilize HTML class awareness to strip non-content elements. • **Optimization:** Achieved a **63% reduction in lines** and **36% reduction in characters**, directly saving compute costs.

	• **Logic:** Implemented an "Accreditation Block" to preserve author/source metadata while cleaning body text.
RAG Database	• **Database:** Migrated from MySQL to **PostgreSQL with pgvector** (IVFFlat indexing). • **Models:** Standardized on **Qwen2.5:0.5b-instruct** for chat and **all-minilm:33m** for embeddings to balance performance with local resource limits. • **Architecture:** Fully Dockerized the service to ensure reproducibility.
Fine Tuning	• **Optimization:** Implemented **SINQ (Quantization)** logic to allow heavy models to run on student laptops. • **Implementation:** Moved from EconBERTa to a **BERT** implementation focused on the SQuAD database. • **Workflow:** Created a Kanban-style automation dashboard to track training jobs.

3. Technical Architecture & Commonality Analysis

Analysis of high-level components based on final project outcomes.

Commonalities (Strengths)

- **Language:** All five teams standardized on **Python** for backend logic.
- **Frontend:** Convergence on a **Unified React UI** ensured a consistent user experience.
- **Local Focus:** All teams optimized for local execution (e.g., RAG selecting 0.5b parameter models; Fine Tuning using SINQ).

Divergences (Friction Points)

- **Databases:** RAG moved to **PostgreSQL**, while Fine Tuning prepares SQuAD data independently.
- **Environment:** RAG is **Docker-based**, while other teams rely on local `requirements.txt`. RAG documentation notes that "Containerization... may limit some future development" if not managed carefully across teams.

- **Data Interfaces:** The Web Scraping team outputs JSON, but the RAG team requires specific chunking formats that are not yet fully aligned.

4. Priorities for Future Classes (The Handoff)

Specific recommendations derived from the "Future Recommendations" section of each team's final report.

Priority 1: The "Smart Router" (Web Scraping Recommendation)

- **Objective:** Automate input handling.
- **Deliverable:** Build a "Smart Router" that recognizes pasted content (Text vs. CSV vs. URL vs. Reddit Link) and automatically dispatches it to the correct processor without user selection.

Priority 2: Advanced RAG Chunking (RAG Recommendation)

- **Objective:** Improve retrieval quality.
- **Deliverable:** Move beyond simple paragraph splits. Implement **Sentence-Window Chunking** and metadata filtering.
- **Action:** RAG team explicitly requests collaboration with Scraping/Cleaning to standardize the JSON input format for these chunks.

Priority 3: UI Implementation Fidelity (Design Team Recommendation)

- **Objective:** Ensure code matches design.
- **Action:** Future developers must utilize the provided **Figma Layout** as the strict "source of truth." Do not code ad-hoc styles; follow the grid and typography defined in the prototype.

Priority 4: Accreditation & Metadata (Data Cleaning Recommendation)

- **Objective:** Preserve data provenance.
- **Deliverable:** Ensure the "Accreditation Block" logic (created by the Cleaning team) is preserved and visualized in the final RAG citation output.

Priority 5: User-Friendly Entry (Fine Tuning Recommendation)

- **Objective:** Lower the barrier to entry.
- **Deliverable:** Implement specific UI graphs below the log terminal to visualize training loss/accuracy in real-time for non-technical users (Business majors).

5. Iteration Schedule Summary

- **Sprint 1:** Architecture & Research (Pivots from Amazon/Twitter; RAG explores Vector DBs).
- **Sprint 2:** Prototyping (Figma Design formed; Agentic creates BPMN).
- **Sprint 3:** Core Development (FastAPI, Content-Aware Algorithms, BERT implementation).
- **Sprint 4:** Breakthroughs (MCP Integration, pgvector migration, 36% noise reduction).
- **Sprint 5:** Integration & Polish (React Frontend integration, Dockerization of RAG, Final Documentation).

6. Program Board (Dependencies)

Dependent Team	Needs...	From...	Status
All Teams	Unified UI Components & Design System	**UI/UX Sub-team**	[Resolved]
Data Cleaning	Raw Data Output (JSON/Text)	**Web Scraping**	[Resolved]
RAG	Defined Data Input Format (Standardized JSON)	**Data Cleaning**	[Critical Priority]
Agentic Systems	Functional RAG API to query docs	**RAG Database**	[At Risk]
All Teams	Unified Docker Environment	**DevOps/Arch**	[Blocked]

7. ROAM Board (Risk Management)

Risk Description	Impact	Category	Mitigation Strategy
Environment Inconsistency: `requirements.txt` differs per machine.	Critical	**Owned**	**Future Action:** Mandate Docker for all teams in Sprint 1 of next semester (RAG team has provided the template).
Containerization Limits: RAG team notes full containerization is resource-heavy.	High	**Accepted**	**Mitigation:** Use a hybrid approach where only DB/API are containerized, and ML training runs on bare metal if GPU is required.
Knowledge Transfer: Complexity of LangFlow/n8n concepts.	Medium	**Mitigated**	**Action:** The "Agent Creator" tool abstracts this complexity, reducing the learning curve for new users.
Ambiguity: Lack of clear task definition (Cleaning Team).	Medium	**Resolved**	**Lesson:** Use BPMN and visual workflows (like Agentic team did) to align goals early in Sprint 1.

8. Strategic Roadmap Alignment (Year 1)

Mapping this PI to the 5-Year Vision.

Strategic Theme	Year 1 Goal (Roadmap)	Status
Ease of Use	Install, bug issues, GUI basics	[Mixed] GUI is **Live & Unified**, but installation remains complex.
RAG Database	Vector DB Integration	[Done] Successfully migrated MySQL -> Postgres/pgvector.
Data Cleaning	Basic cleaning scripts & processes	[Done] Content-aware scripts operational (36% reduction).
Agentic Systems	Prototype basic agent tool	[Done] "LMForge Agent Creator" with MCP is live.