Ouro: Personalized Chatbot with Dynamic Responses

Final Technical Project Report
Project ID: Generative AI - Project 2
Team Lead: Pierre Ramez
Date: November 27, 2025
Status: Production Ready

1. Executive Summary

The **Ouro Project** was engineered to address the limitations of static Large Language Models (LLMs) in personalized environments. While traditional chatbots require expensive, periodic retraining to update knowledge or correct behaviors, Ouro implements a **Closed-Loop Active Learning System**.
This system leverages a decoupled microservices architecture to enable a **Human-in-the-Loop (HITL)** feedback mechanism. By utilizing **Low-Rank Adaptation (LoRA)** and **4-bit Quantization** via the Unsloth framework, we achieved a high-performance conversational agent (Llama 3.2 3B) capable of running on resource-constrained edge infrastructure (CPU/T4 GPU) while maintaining the ability to "heal" its own hallucinations through real-time user corrections.

Milestone 1: Data Collection and Preprocessing

Objective: To curate a high-quality, heterogeneous dataset that balances general conversational fluency with specific emotional intelligence capabilities.

1.1 Data Sourcing Strategy

We aggregated data from three distinct sources to create a robust training corpus:

1. **Daily Dialogue (CSV):** A large-scale dataset (13,118 rows) containing multi-turn conversations about daily life, providing the "chitchat" backbone.
2. **Mental Health Dataset (JSON):** 661 intent-response pairs focused on emotional support and empathy.
3. **Human Chat (TXT):** 94 raw logs of human-to-human interaction to capture realistic turn-taking dynamics.

1.2 The Preprocessing Pipeline

A custom ETL (Extract, Transform, Load) pipeline was engineered using *pandas* and *ftfy* to sanitize the raw inputs.

Key Transformation Steps:

- **Encoding Normalization:** Utilized *ftfy* to fix "mojibake" (broken unicode characters) and standardize text encoding.
- **Noise Reduction:** Removed dataset-specific metadata columns (e.g., act, emotion from Daily Dialogue) to focus purely on semantic content.
- **Regex Cleaning:**
    - Collapsed extraneous whitespace.
    - Standardized punctuation (e.g., converting "Hello ." $\to$ "Hello.").
    - Removed artifact quotes surrounding dialogue turns.

1.3 Final Dataset Schema

The heterogeneous sources were unified into the industry-standard **Llama-3 Chat Format**:

- **Total Volume:** 13,502 high-quality conversational pairs.
- **Format:** JSONL (JSON Lines).
- **Structure:**

```
{
  "messages": [
    {"role": "user", "content": "I feel overwhelmed today."},
    {"role": "assistant", "content": "I understand. Let's take it one step at a time."}
  ]
}
```

Milestone 2: Model Development and Training

Objective: To fine-tune a Transformer-based model that balances reasoning capability with inference latency constraints.

2.1 Comparative Architecture Analysis

We conducted a rigorous A/B test between two architectures to determine the optimal production model.

Candidate A: *Llama 3.1 8B* (The "Reasoning" Model)

- **Configuration:** LoRA Rank 16, Alpha 16, Flash Attention 2.
- **Performance:** High reasoning capability.
- **Constraint:** Required ~14GB VRAM for inference.
- **Outcome: Rejected** for production due to memory constraints on the target deployment environment (Hugging Face Spaces Free Tier supports only 16GB CPU RAM).

Candidate B: *Llama 3.2 3B* (The "Edge" Model)

- **Configuration:** 4-bit NF4 Quantization, LoRA Rank 16.
- **Performance:** Excellent conversational coherence with significantly lower footprint.
- **Constraint:** Fits comfortably within 16GB CPU RAM.
- **Outcome: Selected** for production.

2.2 The Training Pipeline (*Unsloth* Optimization)

We utilized **Unsloth**, a framework that manually optimizes backpropagation steps, allowing for 2x faster training and 60% memory reduction compared to standard Hugging Face implementations.

Hyperparameter Configuration:

Parameter	Value	Justification
Precision	bf16 (Mixed)	Balances numerical stability with memory usage.
Quantization	4-bit (NF4)	Reduces VRAM usage from ~6GB to ~2.5GB.
LoRA Rank (r)	16	Sufficient plasticity for style transfer without overfitting.
Optimizer	adamw_8bit	Further reduces optimizer state memory footprint.
Max Seq Length	2048	Covers standard chat history context windows.

2.3 Evaluation Strategy: The Sliding Window

We rejected standard "final-turn" evaluation in favor of a **Sliding** Window **Strategy**. This involves taking a multi-turn conversation and evaluating the model's prediction accuracy at *every* assistant turn, not just the end.

Quantitative Results:

- **Perplexity:** 4.08 (Indicates high predictive confidence).
- **BERTScore F1: 0.8769** (Indicates ~88% semantic alignment with human references).
- **Precision:** 0.8849 | **Recall:** 0.8693.

Milestone 3: Advanced Techniques (Continuous Learning)

Objective: To architect a "Self-Healing" AI pipeline that integrates user corrections.

This milestone represents the project's core innovation. Instead of a static model, Ouro implements a dynamic feedback loop.

3.1 The Feedback Architecture

1. **Interaction:** User chats with the bot via the Frontend.
2. **Intervention:** The user identifies an error (hallucination) and submits a "Correction" via the UI.
3. **Ingestion:** The backend captures this correction into a persistent feedback.jsonl ledger.
4. **Orchestration:** A custom PipelineRun class triggers the learning workflow:
    - **Data Preparation:** Converts the raw correction into a training batch.
    - **Active Training:** Initiates a targeted *SFT* run on the specific correction data.
    - **Validation:** Checks if the new adapter passes "Quality Gates" (Semantic Similarity > 0.85).
    - **Hot-Swap:** The system unloads the old LoRA adapter and loads the new one *without* restarting the server.

3.2 Dynamic Adapter Management

By leveraging *PeftModel*, we decoupled the base model weights (frozen) from the behavior weights (adapters). This allows us to "patch" the model's behavior in seconds rather than hours.

Milestone 4: MLOps and Model Management

Objective: To ensure reproducibility, traceability, and safe deployment.

4.1 Experiment Tracking (MLflow)

We integrated ***MLflow*** (hosted via *DagsHub*) to track every training run.

- **Metrics Logged:** Loss curves, Learning Rate, ROUGE-L, BLEU, BERTScore.
- **Artifacts:** The trained adapter files and tokenizer configurations are versioned and stored.
- **Registry:** Only models that pass the automated Quality Gates are tagged as production_candidate.

4.2 CI/CD Pipeline (GitHub Actions)

We implemented a robust Continuous Integration/Continuous Deployment pipeline defined in *main.yml*.

Workflow:

1. **Trigger:** Code push to main branch.
2. **Build & Test:**
    - Sets up Python environment.
    - Installs dependencies.
    - Runs *pytest* on *test_app.py* to verify API endpoints (/health, /chat, /feedback).
3. **Deploy:**
    - If tests pass, the code is pushed to the **Hugging Face Space**.
    - The Docker container is automatically rebuilt and redeployed.

4.3 Containerization

The application is packaged using **Docker** to ensure consistency across environments (Colab, Local, Cloud).

- **Permissions:** Configured *chmod* 777 on the /data directory to ensure the application can write feedback logs even in restricted serverless environments.

Milestone 5: Final Deployment

Objective: To deliver a user-friendly, accessible product.

5.1 Deployment Stack

- **Frontend:** Hugging Face Spaces (*Streamlit*). Provides a chat interface with a custom "Spotted a mistake?" expander.
- **Backend:** Hugging Face Spaces (*FastAPI*). Handles inference, queueing, and feedback logic.
- **Model Registry:** Hugging Face Hub. Stores the *Llama 3.2 3B* base model and the fine-tuned LoRA adapters.

5.2 Zero-Downtime Updates

The deployment architecture supports **Hot Reloading**. When a new adapter is trained:

1. The pipeline pushes the adapter to the Hub.
2. The backend receives a POST /reload-adapter request.
3. The inference engine swaps the weights in memory immediately.
4. The user experiences improved performance instantly.

5.3 Links & Resources

- **GitHub Repo:** PierreRamez/PersonalChatbot
- **Production Model:** pierreramez/Llama-3.2-3B-Instruct-bnb-4bit_finetuned
- **Live Demo:** Chatbot UI
- **MLflow Tracking:** DagsHub Project

Conclusion

The **Ouro Project** successfully transcends the typical limitations of academic AI projects. By integrating a **Continuous** Learning **Loop**, we have moved beyond creating a "smart" chatbot to creating a "learning" chatbot. The combination of ***Unsloth*** for efficient training, **LoRA** for modular updates, and a **CI/CD** pipeline for safe deployment results in a resilient