SPARC-P: Standardized Patient Assessment & Remediation System (HiPerGator Notebooks)

Quick Links

- **API Documentation** - Consolidated function/class/endpoint reference
- **Migration Guide** - ⚠ **READ THIS FIRST** - Updated to use conda (UF RC requirement)
- **PubApp Deployment Guide** - Deploy trained models for public access

---

⚠ Important Update (February 2026)

This repository has been updated to follow UF RC best practices:

What Changed?

- ☑ **Now using conda** instead of pip for package management (UF RC requirement)
- ☑ **Added PubApp deployment notebook** for public model serving
- ☑ **Updated SLURM scripts** to use conda environments
- ☑ **Better CUDA integration** and performance

Quick Start (New Workflow)

```
# 1. One-time setup: Create conda environment on HiPerGator
cd /path/to/Sparc\ Hipergator\ Notebooks
bash setup_conda_env.sh both

# 2. In SLURM scripts or interactive sessions:
module load conda
conda activate /blue/jasondeanarnold/SPARCP/conda_envs/sparc_training

# 3. Run notebooks or scripts as usual
python train_agent.py
```

See MIGRATION_GUIDE.md for complete migration instructions.

Overview

SPARC-P is a digital human training platform designed to help clinicians practice vaccine communication skills. This project implements the backend infrastructure, agent training pipeline, and deployment strategy for the SPARC-P system on the University of Florida's HiPerGator AI SuperPOD.

The system employs a **Hybrid RAG (Retrieval-Augmented Generation) and Fine-Tuning** architecture to create specialized agents:

- **Caregiver Agent**: Simulates a hesitant parent/caregiver with specific personas and emotional responses.
- **Coach Agent**: Evaluates the clinician's performance using the C-LEAR rubric and provides feedback.
- **Supervisor Agent**: Orchestrates the conversation, ensures safety (via NeMo Guardrails), and routes messages.

System Architecture

The SPARC-P backend is designed for high-performance computing (HPC) environments and strict data compliance.

- **Compute**:
  - **Training**: HiPerGator AI SuperPOD (NVIDIA A100/B200 GPUs)
  - **Production**: PubApps (NVIDIA L4 GPUs for inference)
- **Environment Management**: **Conda** (UF RC requirement - see Migration Guide)
- **Containerization**:
  - **HiPerGator**: Apptainer/Singularity
  - **PubApps**: Podman (NO Docker support)
- **Storage**:
  - **HiPerGator**: `/blue` tier for large datasets and models
  - **PubApps**: `/pubapps` tier (1TB included per instance)
- **Orchestration**: LangGraph for managing the multi-agent state machine
- **Speech Services**: NVIDIA Riva for Automatic Speech Recognition (ASR) and Text-to-Speech (TTS)
- **Safety**: NVIDIA NeMo Guardrails for content filtering and topic adherence
- **Deployment**: Systemd services on PubApps for persistent backend hosting

---

Repository Structure

```
Sparc Hipergator Notebooks/
├── README.md                                # This file
├── API_DOCUMENTATION.md                     # API reference for all notebooks
├── MIGRATION_GUIDE.md                       # ⚠️ Conda migration instructions
│
├── environment_training.yml                 # Conda env for training (HiPerGator)
├── environment_backend.yml                  # Conda env for backend
(HiPerGator/PubApps)
├── setup_conda_env.sh                       # Automated conda setup script
│
├── 1_SPARC_Agent_Training.md                # Training pipeline
├── 1_SPARC_Agent_Training.ipynb
├── 2_SPARC_Containerization_and_Deployment.md  # Container/deployment
├── 2_SPARC_Containerization_and_Deployment.ipynb
├── 3_SPARC_RIVA_Backend.md                  # Real-time backend
├── 3_SPARC_RIVA_Backend.ipynb
├── 4_SPARC_PubApp_Deployment.md             # 🆕 PubApp deployment guide
│
├── images/                                   # Architecture diagrams
```

```
├── training_data/                    # Training datasets
└── trained_models/                   # Output models
```

Workflow: Training → Deployment

Phase 1: Training on HiPerGator (Notebooks 1-3)

1. **Notebook 1**: Train agent models using QLoRA on A100/B200 GPUs
2. **Notebook 2**: (Optional) Containerize for complex dependencies
3. **Notebook 3**: Test backend services on HiPerGator

Phase 2: Deployment to PubApps (Notebook 4)

4. **Notebook 4**: Deploy trained models to PubApps for public access
    - Transfer models from HiPerGator `/blue` to PubApps `/pubapps`
    - Set up conda environment on PubApps VM
    - Deploy Riva speech services with Podman
    - Create FastAPI backend with systemd
    - Configure NGINX reverse proxy + UF Shibboleth SSO

Key Distinction:

- **HiPerGator** = Training intensive models
- **PubApps** = Serving models to public via web interface

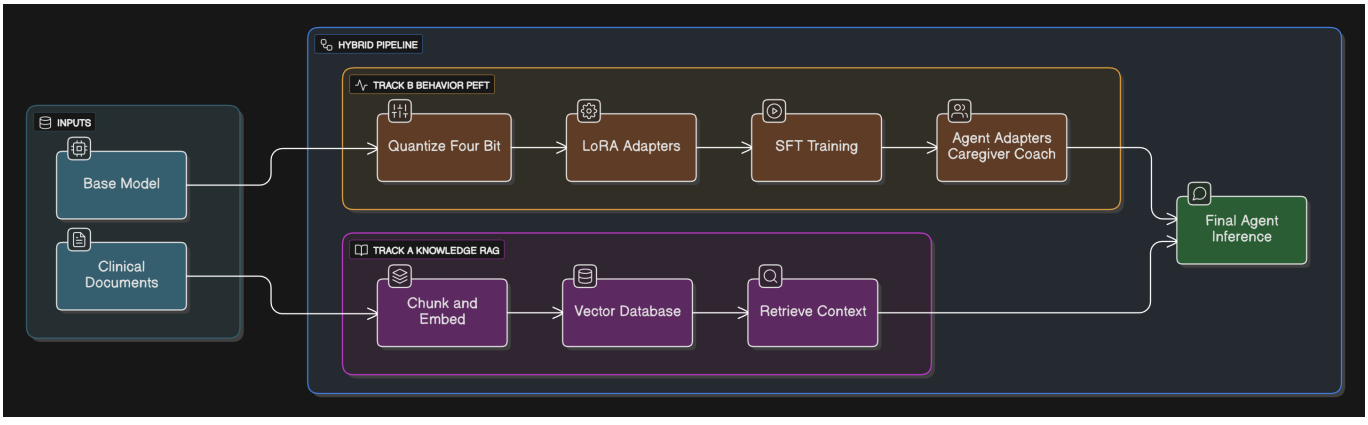
Project Notebooks

The project is documented and implemented across three primary Jupyter notebooks. Below is a detailed breakdown of each notebook with architectural diagrams.

1. Agent Training (`1_SPARC_Agent_Training.ipynb`)

This notebook implements the training pipeline for the SPARC-P agents.

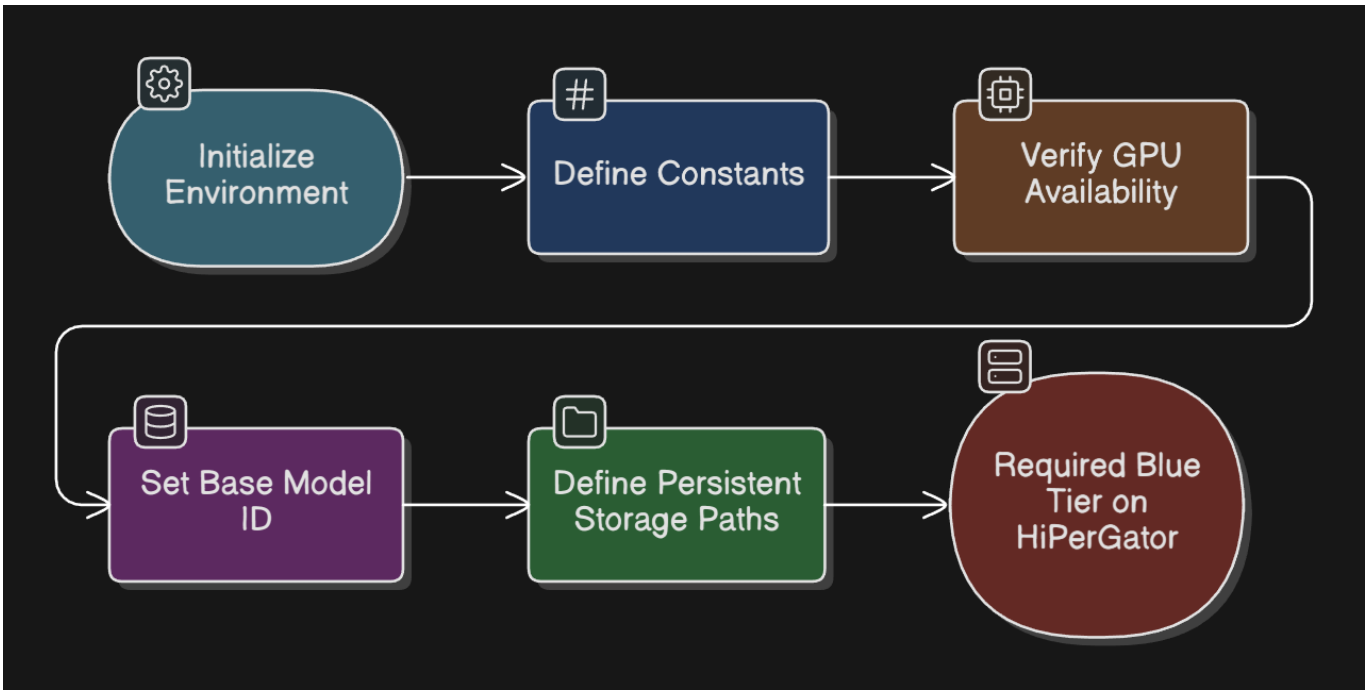
1.1 Introduction and System Purpose



Description: This diagram illustrates the hybrid architecture used in this notebook. It shows how the system

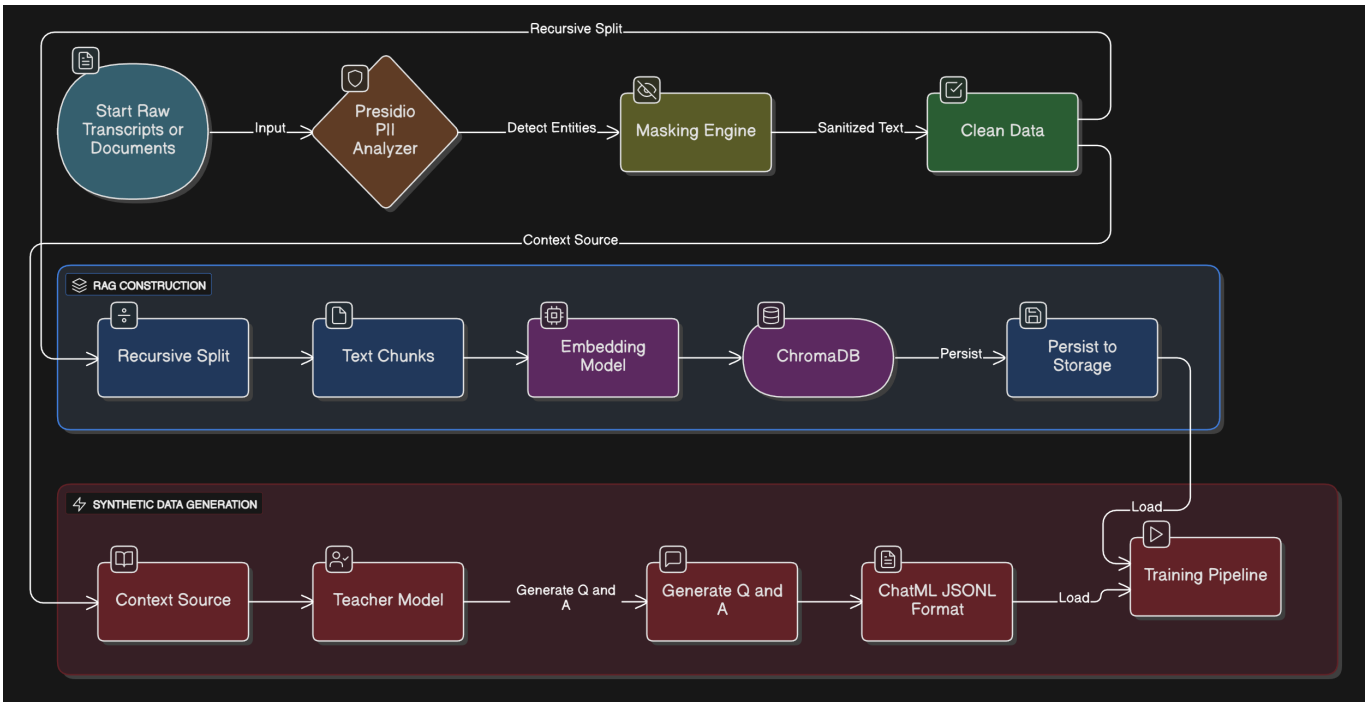
splits into two parallel tracks: RAG (Retrieval-Augmented Generation) for factual grounding using vector databases, and PEFT (Parameter-Efficient Fine-Tuning) using QLoRA to adapt the base model's style and behavior to specific personas (Caregiver, Coach, Supervisor).

1.2 System Configuration



Description: This section initializes the environment settings on HiPerGator. It defines constants, verifies GPU availability, sets the base model ID (gpt-oss-120b), and crucially defines the persistent storage paths on the /blue storage tier, which is required for handling large-scale datasets that exceed standard home directory limits.

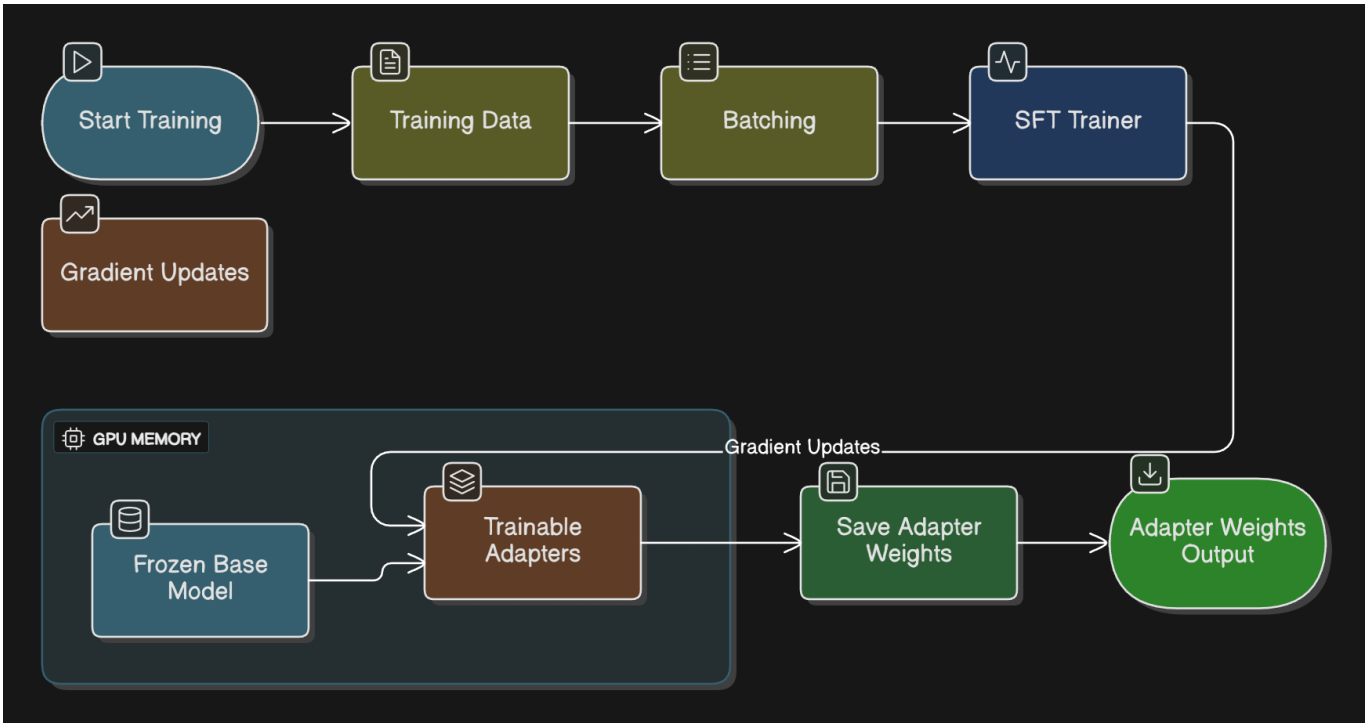
1.3 Data Pipeline



Description: This section covers the data preparation lifecycle. Raw clinical text is first passed through Microsoft Presidio to strip Personally Identifiable Information (PII). The sanitized text is then split: one path

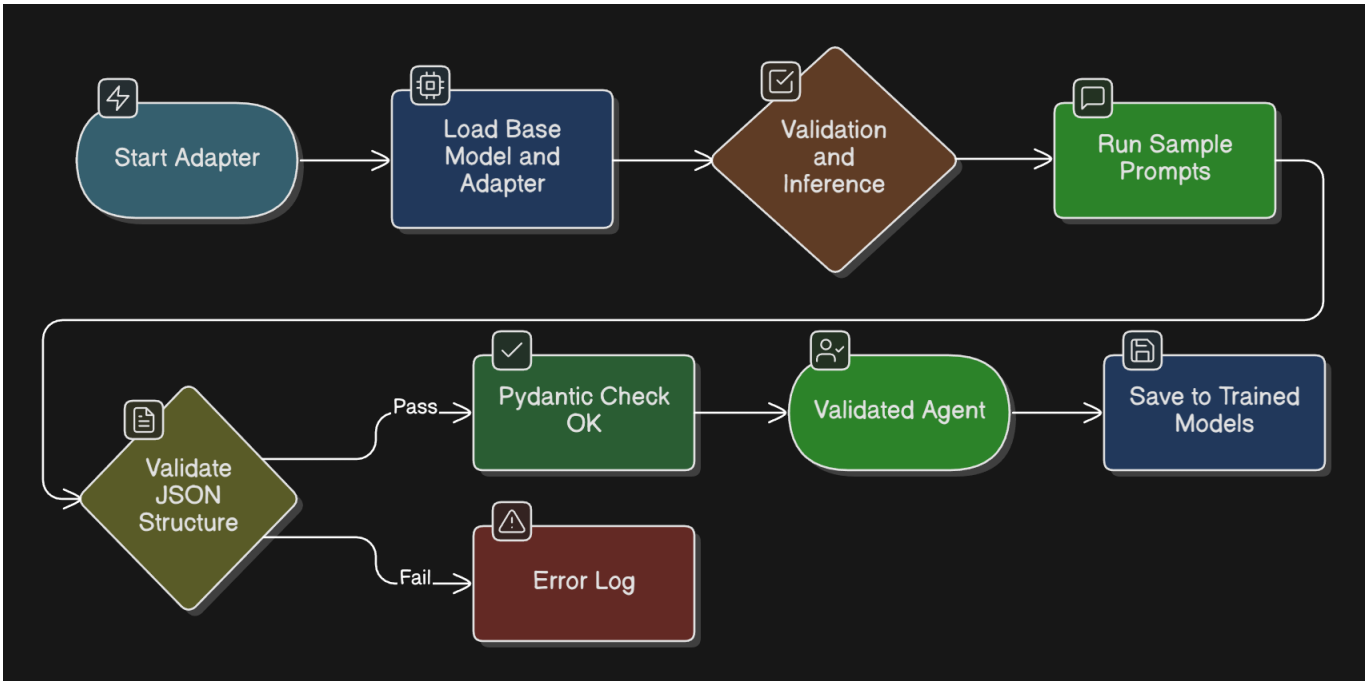
builds the RAG Vector Store (ChromaDB) for factual queries, while the other uses a "Teacher Model" to generate synthetic question-answer pairs for fine-tuning.

1.4 Fine-Tuning (QLoRA)



Description: This diagram visualizes the QLoRA training loop. It highlights how the massive base model is frozen and quantized to 4-bit precision to fit on the GPU. Small, trainable "Adapter" layers are attached to the attention modules. The SFTTrainer updates only these adapters based on the synthetic dataset, resulting in a lightweight, portable model file.

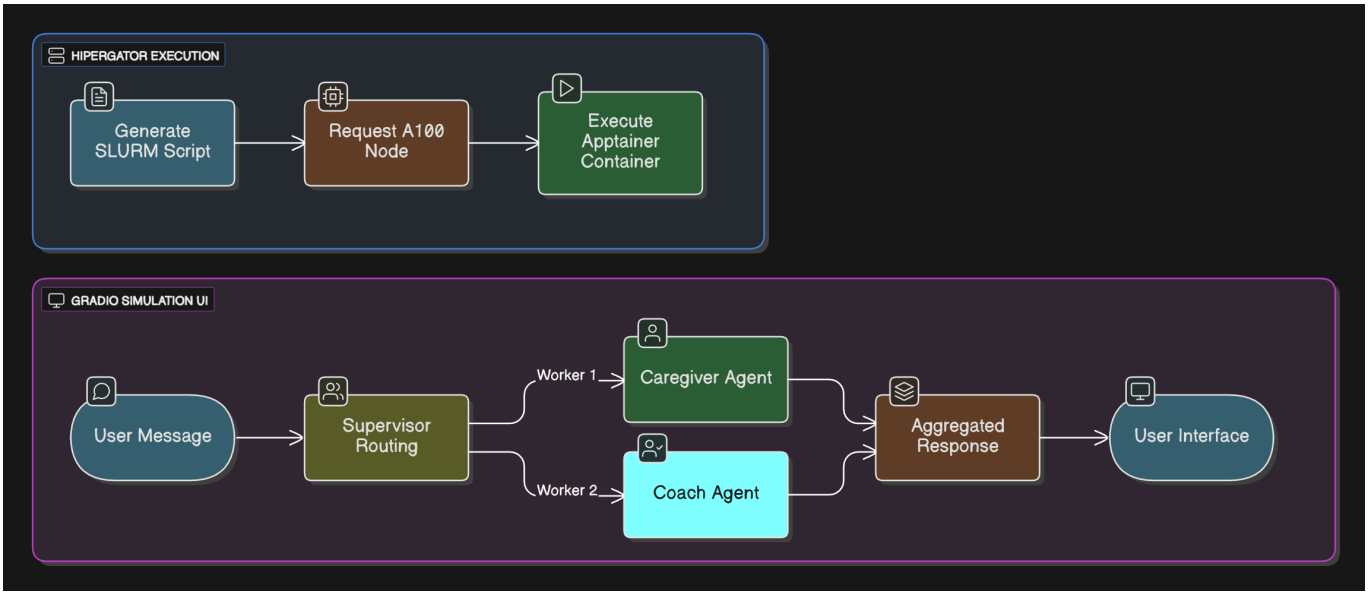
1.5 Validation



Description: After training, the system must validate that the agents produce valid outputs. This workflow loads the base model combined with the new adapter, runs sample inference prompts, and uses Pydantic

schemas to validate the structure of the JSON output (e.g., checking for specific fields like emotion or grade) before saving the final adapters.

1.6 Interfaces



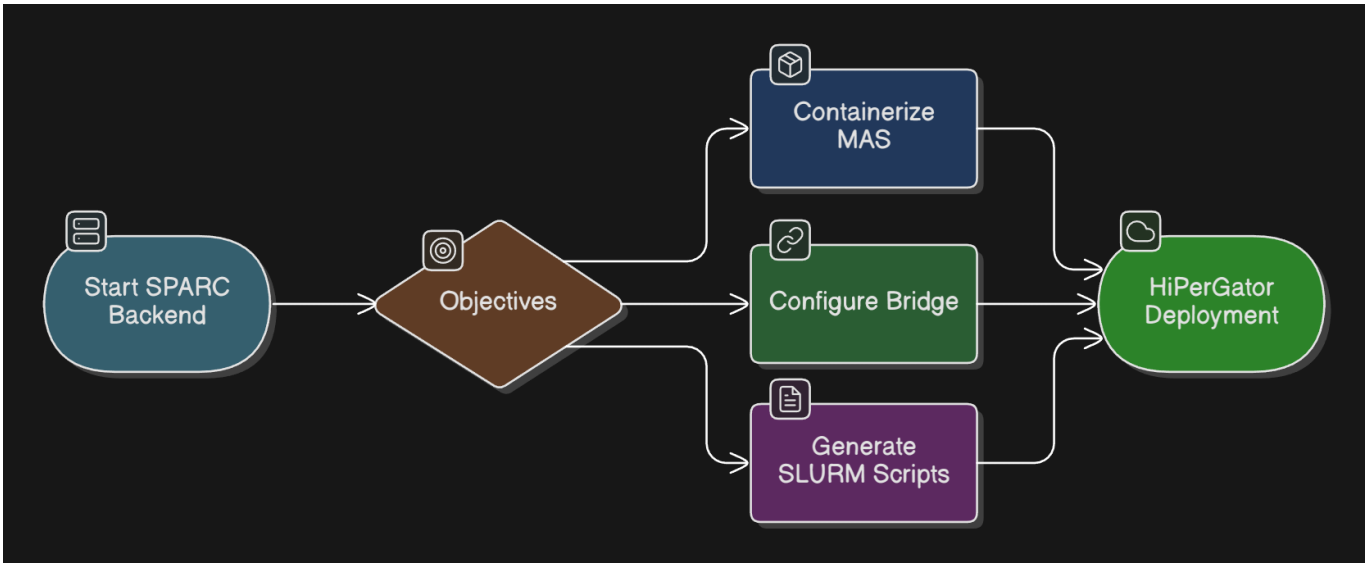
Description: This section covers the final testing and submission interfaces. It generates a SLURM script to run the training job on a GPU node via Apptainer. It also includes a Gradio interface that simulates the full multi-agent loop, showing how the Supervisor routes messages to the Caregiver or Coach and aggregates the response.

---

2. Containerization & Deployment (`2_SPARC_Containerization_and_Deployment.ipynb`)

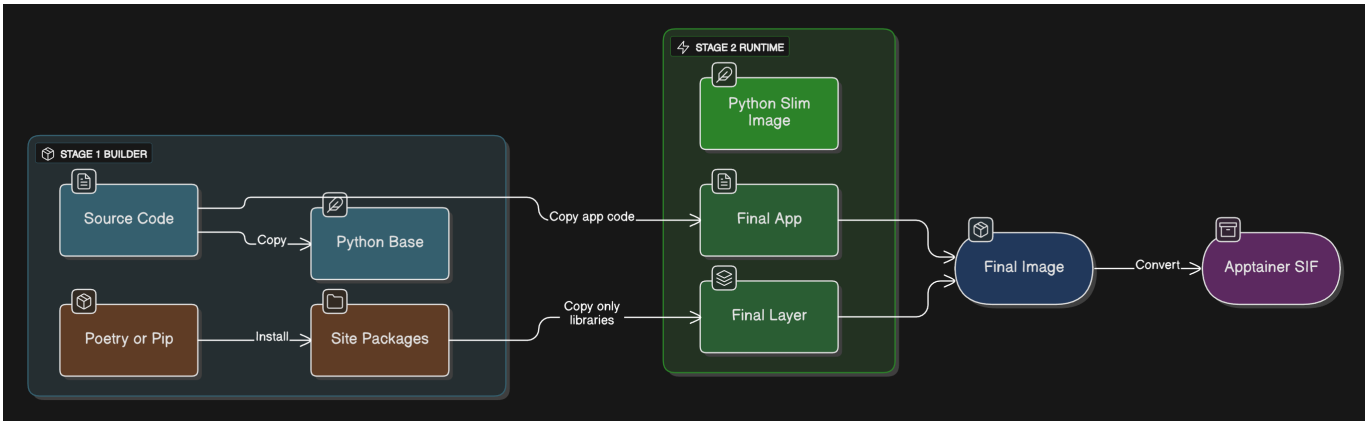
This notebook handles the packaging and deployment of the backend system.

2.1 Objectives



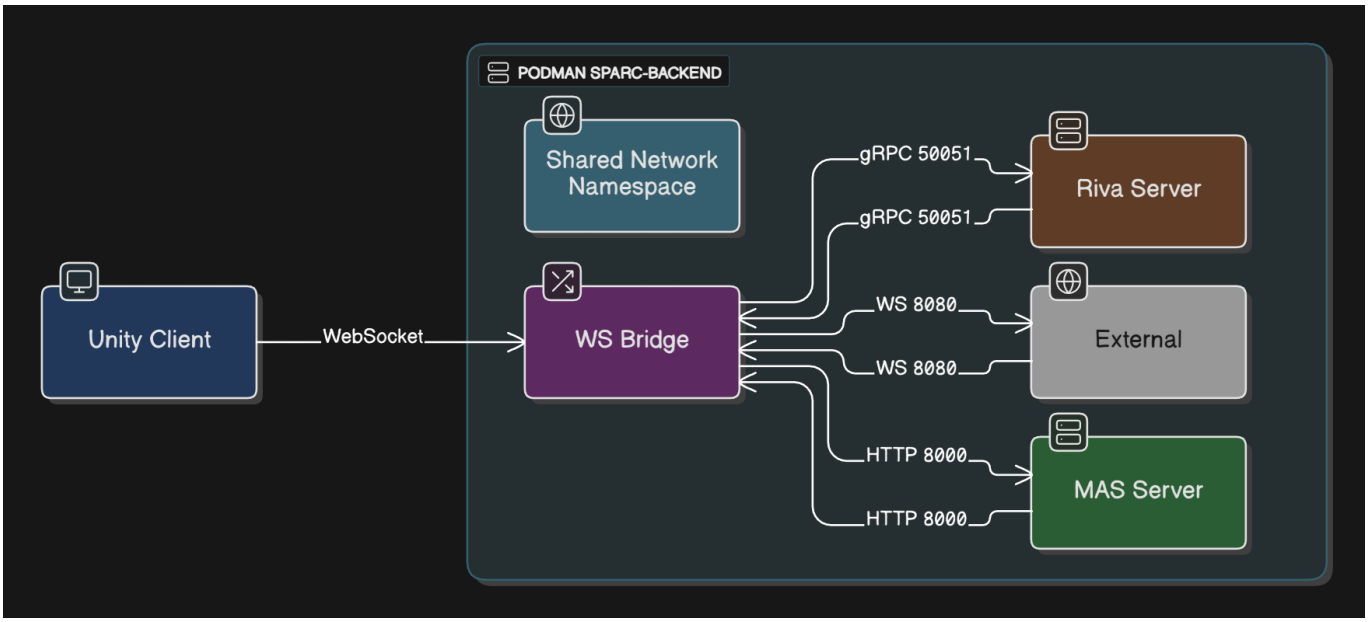
Description: This section sets the objectives for packaging and deploying the backend. The goal is to containerize the Multi-Agent System (MAS), configure the WebSocket-to-gRPC bridge for Unity connectivity, and generate robust production SLURM scripts for deployment to HiPerGator.

2.2 Container Build Strategy



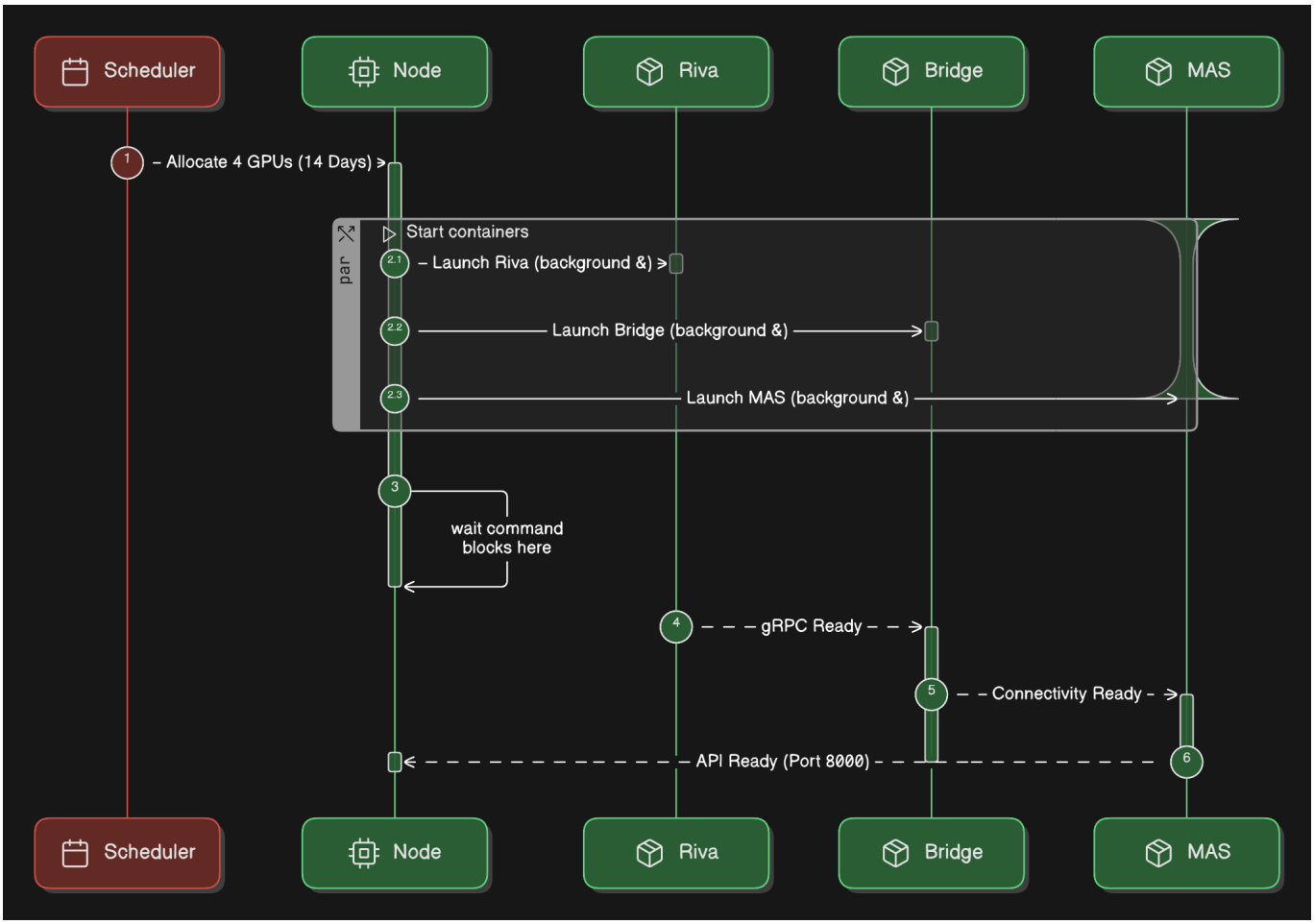
Description: This flow shows the Multi-Stage Build strategy used to create secure and small containers. A "Builder" stage uses Poetry to compile dependencies, and then only the necessary artifacts are copied over to a slim "Runtime" stage. This excludes compiler tools and cache files from the final production image.

2.3 Local Development (Podman)



Description: This illustrates the local development environment using Podman Pods. Unlike standard Docker containers which are isolated, a "Pod" shares a network namespace (localhost). This allows the Riva Server, WebSocket Bridge, and MAS (Multi-Agent System) to communicate locally, perfectly simulating the production environment on a developer's machine.

2.4 Production Deployment



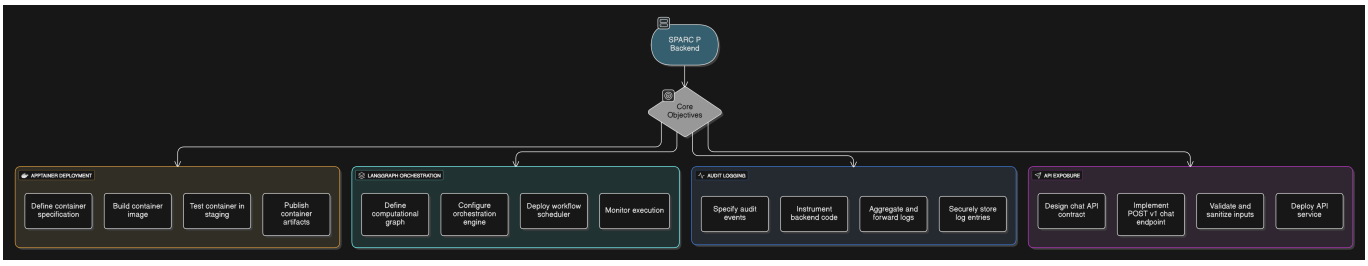
Description: This diagram shows the execution flow of the sparc_production.slurm script on HiPerGator. It details how the SLURM scheduler allocates resources (GPUs) and then launches three concurrent Apptainer containers in the background, keeping them alive with a wait command.

---

3. Real-Time Backend (`3_SPARC_RIVA_Backend.ipynb`)

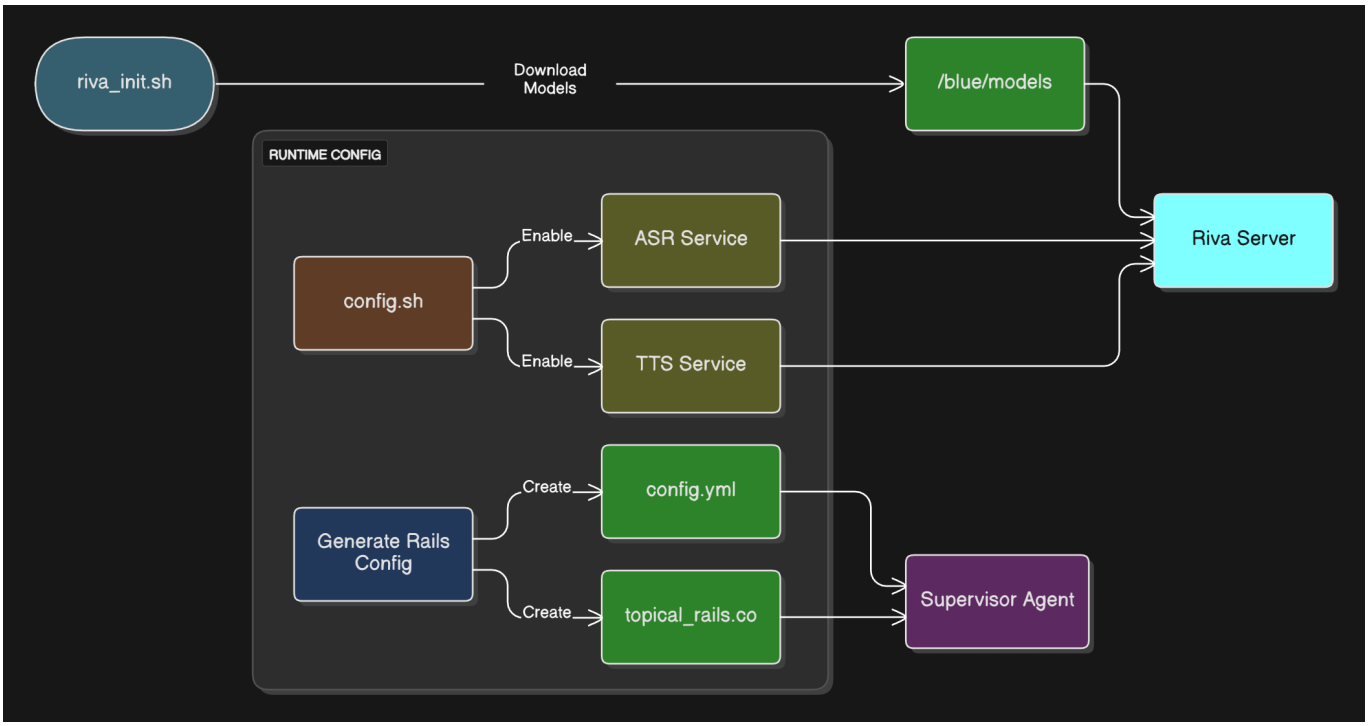
This notebook implements the runtime execution of the backend.

3.1 Runtime Objectives



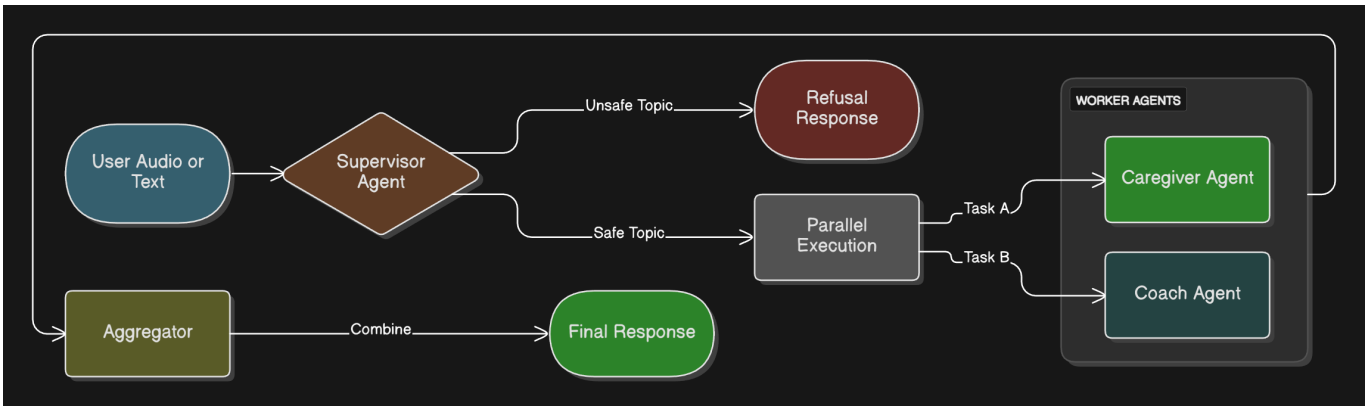
Description: This section defines the objectives for the real-time backend. It implements the Real-Time, Multi-Agent Backend on HiPerGator, utilizing Apptainer for containerization, LangGraph for orchestration, and immutable audit logging to the /blue tier for compliance.

3.2 Riva & Guardrails



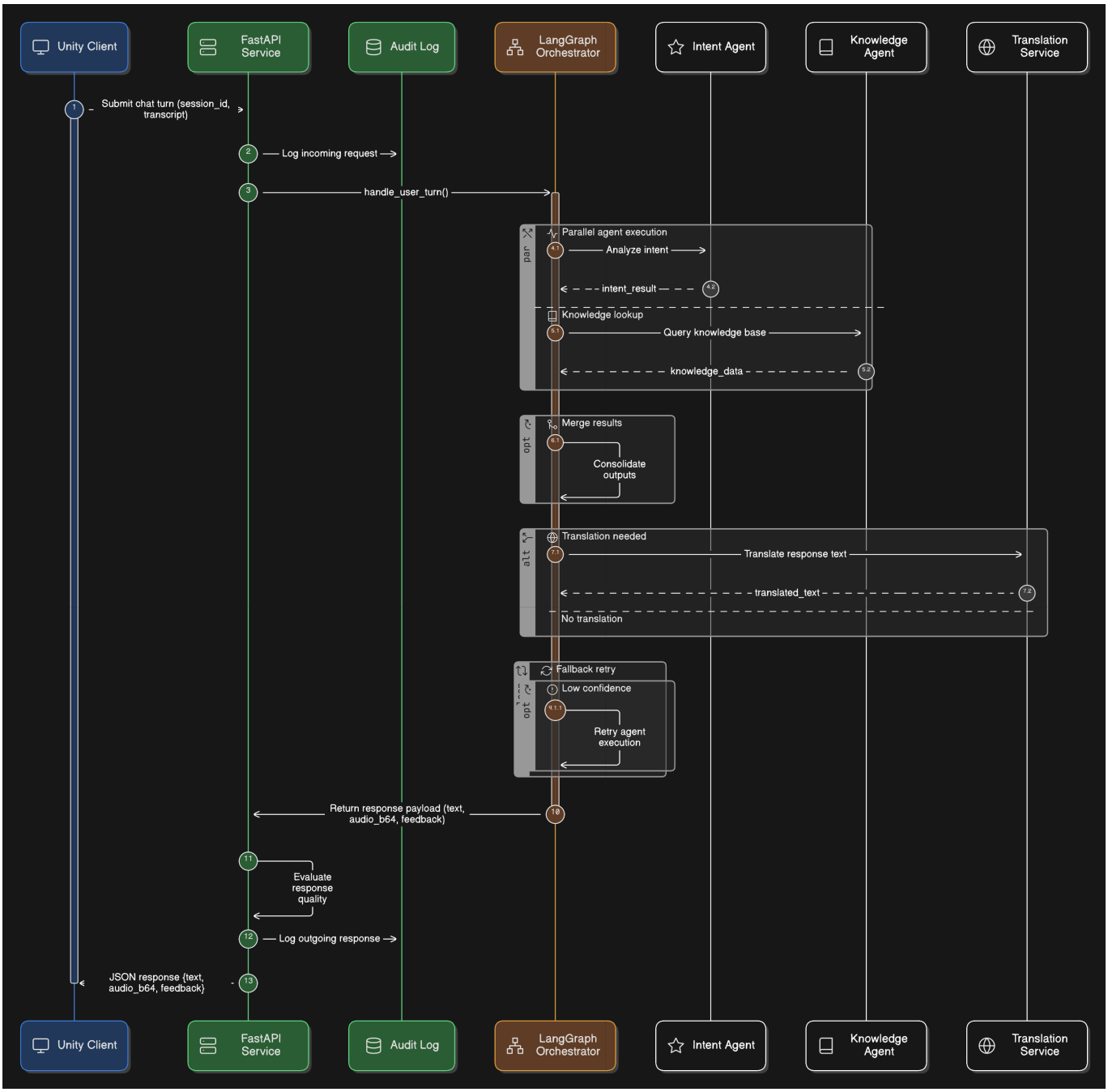
Description: This chart depicts the initialization of the speech services and safety rails. The Riva server is initialized with ASR (Speech-to-Text) and TTS (Text-to-Speech) enabled. Concurrently, NeMo Guardrails configuration files (config.yml, topical_rails.co) are generated to define the "boundary" of the conversation (e.g., refusing political topics).

3.3 Multi-Agent Orchestration



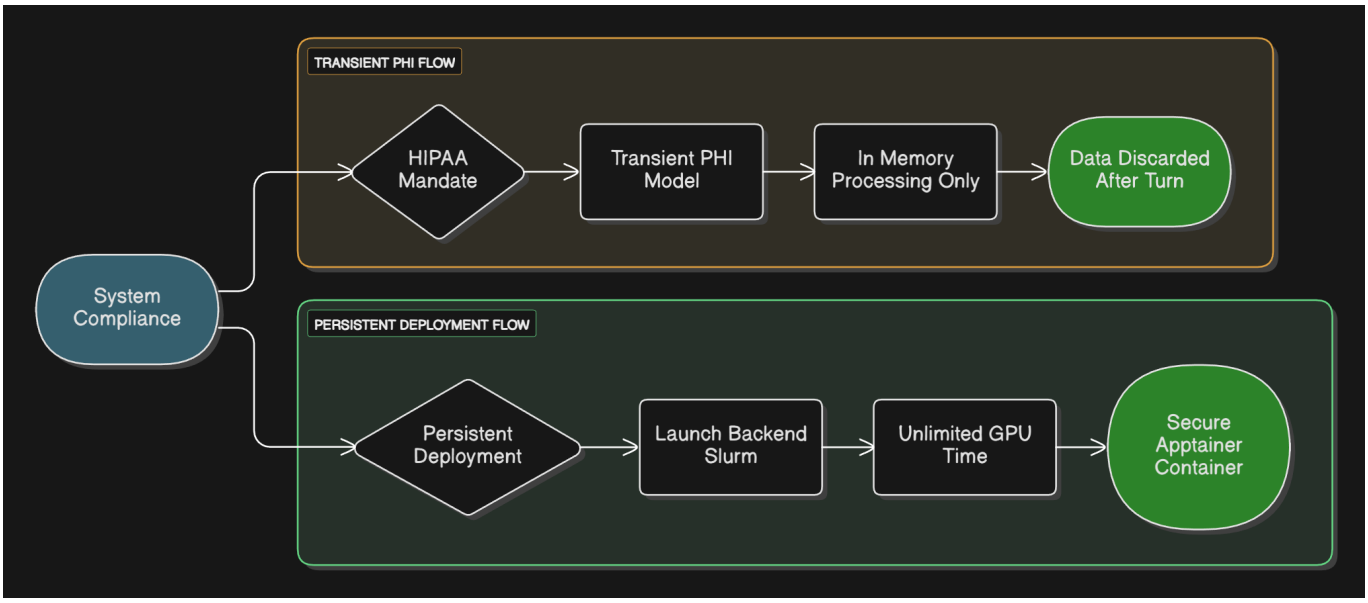
Description: This is the core logic of the backend. It visualizes the Supervisor-Worker pattern. The User Input is first checked by the Supervisor (Guardrails). If safe, it triggers the Caregiver (generating the response) and the Coach (evaluating the response) in parallel to minimize latency. The results are aggregated into a single JSON response.

3.4 API Server Integration



Description: This diagram maps the data flow through the FastAPI application. The Unity Client sends a request to /v1/chat. The server logs the request for auditing, invokes the LangGraph orchestration loop (defined in Section 5), and returns the structured ChatResponse containing text, audio (Base64), and animation cues.

3.5 Security and Compliance



Description: This section outlines the security protocols and persistent deployment. It adheres to the HIPAA Mandate using a 'Transient PHI' model, where user data is processed in-memory and immediately discarded. The launch_backend.slurm script ensures the service runs persistently on a secure GPU node.

---

4. PubApp Deployment (`4_SPARC_PubApp_Deployment.md`) NEW

This guide covers deploying the trained models to **UF RC PubApps** for public access.

Description: Complete step-by-step guide for:

- Provisioning a PubApps instance ($300/year allocation)
- Transferring trained models from HiPerGator to PubApps
- Setting up conda environment on PubApps VM
- Deploying Riva speech services with Podman containers
- Creating FastAPI backend with systemd service management
- Configuring NGINX reverse proxy with UF Shibboleth SSO
- Monitoring, maintenance, and troubleshooting

Key Sections:

- Prerequisites and instance setup
- Model transfer from HiPerGator
- Conda environment setup on PubApps
- Riva deployment with Podman
- FastAPI backend with systemd
- NGINX configuration and SSL
- Security and compliance (Shibboleth, transient PHI model)
- Monitoring and troubleshooting

Prerequisites

For Training (HiPerGator)

- **Account**: Active HiPerGator account with GPU access

- **Hardware**: Access to GPU-enabled nodes (NVIDIA A100/B200)
- **Software**:
  - Conda (via `module load conda`)
  - CUDA 12.8+ (via `module load cuda/12.8`)
  - Apptainer (for optional containerization)
- **Storage**: Write access to `/blue/jasondeanarnold/SPARCP` directory
- **Allocation**: Sufficient SUs (Service Units) for GPU training

For Deployment (PubApps)

- **Allocation**: PA-Instance ($300/year) + optional PA-GPU for inference
- **Risk Assessment**: Completed via RC Risk Assessment Process
- **SSH Access**: SSH key registered for PubApps instance access
- **Domain**: Assigned domain (e.g., `sparc-p.rc.ufl.edu`)
- **Firebase**: Google Firebase project for session state management

---

Quick Start Guide

Step 0: Environment Setup (One-Time)

```
# SSH to HiPerGator
ssh jayrosen@hpg.rc.ufl.edu

# Clone/download this repository
cd /blue/jasondeanarnold/SPARCP
git clone https://github.com/UF-College-of-Education/SPARCP-Hipergator-
Notebooks.git
cd SPARCP-Hipergator-Notebooks

# Edit setup script with your group name
nano setup_conda_env.sh
# Confirm: GROUP_NAME="jasondeanarnold"

# Run automated setup (creates both environments)
bash setup_conda_env.sh both

# Verify setup
module load conda
conda activate /blue/jasondeanarnold/SPARCP/conda_envs/sparc_training
python -c "import torch; print(f'CUDA: {torch.cuda.is_available()}')"
```

Step 1: Train Agents (Notebook 1)

```
# Activate training environment
module load conda
conda activate /blue/jasondeanarnold/SPARCP/conda_envs/sparc_training
```

```
# Run training notebook or generate SLURM script
jupyter notebook 1_SPARC_Agent_Training.ipynb

# OR submit via SLURM:
# (First generate script via notebook Section 6.4)
sbatch train_agent.slurm
```

Step 2: (Optional) Containerize (Notebook 2)

Note: Containers are optional on HiPerGator. Conda is preferred for most use cases.

```
# Only needed if you have complex dependencies
# See 2_SPARC_Containerization_and_Deployment.md
```

Step 3: Test Backend (Notebook 3)

```
# Activate backend environment
module load conda
conda activate /blue/jasondeanarnold/SPARCP/conda_envs/sparc_backend

# Run backend notebook
jupyter notebook 3_SPARC_RIVA_Backend.ipynb
```

Step 4: Deploy to PubApps (Notebook 4)

```
# Follow complete guide in:
# 4_SPARC_PubApp_Deployment.md

# Summary:
# 1. Request PubApps instance (support ticket)
# 2. Transfer models: rsync or Globus
# 3. Install conda on PubApps VM
# 4. Deploy Riva + FastAPI backend
# 5. Configure NGINX + Shibboleth
# 6. Test end-to-end with Unity WebGL
```

Prerequisites

- **Hardware**: Access to a GPU-enabled node (e.g., NVIDIA A100).
- **Software**:
  - Python 3.10+
  - Apptainer (Singularity)
  - NVIDIA Riva

- **Storage**: Access to the `/blue` storage tier on HiPerGator.

Usage (Legacy - Pre-Conda)

⚠ This section describes the old workflow. See "Quick Start Guide" above for the current conda-based workflow.

Security & Compliance

- **HIPAA Compliance**: The system is designed to handle 'Transient PHI'. Audio and transcripts are processed in volatile memory and are strictly not logged to disk.
- **Audit Logging**: Non-sensitive operational logs are written to `/blue/jasondeanarnold/SPARCP/logs/audit.log` for compliance tracking.
- **Guardrails**: NeMo Guardrails ensure the agents do not engage in off-topic or unsafe discussions (e.g., politics, medical advice outside scope).
- **Authentication**: UF Shibboleth SSO for PubApps deployment (credentials not handled by application).
- **Data Classification**:
  - **Open Data**: Training materials, public health information
  - **Sensitive Data**: Session metadata (transient, in-memory only)
  - **Not Stored**: PHI, FERPA records, user credentials
- **Compliance Documentation**: See PubApp instance request form and risk assessments

---

Additional Resources

UF RC Documentation

- **Conda on HiPerGator**: https://docs.rc.ufl.edu/software/conda_installing_packages/
- **PubApps Hosting**: https://docs.rc.ufl.edu/services/web_hosting/
- **SLURM Job Submission**: https://docs.rc.ufl.edu/scheduler/
- **GPU Access**: https://docs.rc.ufl.edu/scheduler/gpu_access/

Project Resources

- **Main Unity Project**: See parent directory `SPARC-P/`
- **API Documentation**: API_DOCUMENTATION.md
- **Migration Guide**: MIGRATION_GUIDE.md
- **PubApp Deployment**: 4_SPARC_PubApp_Deployment.md

Support

- **UF RC Support**: https://support.rc.ufl.edu/
- **Project Team**: Jason Arnold (jda@coe.ufl.edu), Jay Rosen (jayrosen@ufl.edu)
- **Principal Investigators**: Carma Bylund (carma.bylund@ufl.edu), Jason Arnold (jda@coe.ufl.edu)

Troubleshooting

Common Issues

Issue: "conda: command not found"

```
# Solution: Load conda module first
module load conda
```

Issue: "CUDA not available" in conda environment

```
# Solution: Load CUDA module and verify installation
module load cuda/12.8
python -c "import torch; print(torch.cuda.is_available())"
```

Issue: "Permission denied" writing to /blue

```
# Solution: Check group membership and directory permissions
id -gn  # Check your group
ls -la /blue/jasondeanarnold/SPARCP/  # Check permissions
```

Issue: Home directory quota exceeded

```
# Solution: Use path-based conda envs on /blue
# Don't use: conda create -n myenv
# Use: conda create -p /blue/jasondeanarnold/SPARCP/conda_envs/myenv
```

For more troubleshooting, see:

- MIGRATION_GUIDE.md - Conda-specific issues
- 4_SPARC_PubApp_Deployment.md - PubApps deployment issues

Version History

v2.0 (February 2026) - Conda Migration + PubApp Deployment

- ☑ Migrated from pip to conda (UF RC requirement)
- ☑ Added `environment_training.yml` and `environment_backend.yml`
- ☑ Added automated setup script (`setup_conda_env.sh`)
- ☑ Updated all notebooks to use conda
- ☑ Updated SLURM scripts for conda activation
- ☑ Added PubApp deployment guide (`4_SPARC_PubApp_Deployment.md`)
- ☑ Added migration guide (`MIGRATION_GUIDE.md`)

- ☑ Better CUDA integration and performance

v1.0 (2025) - Initial Release

- Training pipeline with QLoRA fine-tuning
- Containerization with Docker/Apptainer
- Riva backend for ASR/TTS
- LangGraph orchestration
- Multi-agent system (Supervisor, Coach, Caregiver)

Contributing

This project is part of ongoing research at the University of Florida College of Education. For questions, issues, or contributions, please contact the project team.

License

[Specify license or indicate proprietary/research use]

Citation

If you use this work in research, please cite:

```
[Citation information to be added]
```