

# SPARC Hipergator Notebooks API Documentation

---

## Scope

This document describes the callable API surface defined in the three notebook tracks:

- [1\\_SPARC\\_Agent\\_Training.ipynb](#)
- [2\\_SPARC\\_Containerization\\_and\\_Deployment.ipynb](#)
- [3\\_SPARC\\_RIVA\\_Backend.ipynb](#)

It is based on the notebook companion docs:

- [1\\_SPARC\\_Agent\\_Training.md](#)
- [2\\_SPARC\\_Containerization\\_and\\_Deployment.md](#)
- [3\\_SPARC\\_RIVA\\_Backend.md](#)

Note: Several functions are prototype scaffolds/mocks intended for notebook workflow and architecture validation. They are documented here as notebook APIs, not production-hardened library APIs.

## Table of Contents

1. [Shared Runtime Contracts](#)
  2. [Notebook 1 API: Agent Training](#)
  3. [Notebook 2 API: Containerization & Deployment](#)
  4. [Notebook 3 API: RIVA Backend](#)
  5. [Endpoint Contracts](#)
  6. [Generated Artifacts](#)
  7. [Operational Notes](#)
- 

## Shared Runtime Contracts

Global constants used across notebooks

- `BASE_MODEL_ID = "gpt-oss-120b"`
- `OUTPUT_DIR = "/blue/jasondeanarnold/SPARCP/trained_models/"`
- `DATA_DIR = "/blue/jasondeanarnold/SPARCP/training_data/"`
- `RIVA_VERSION = "2.16.0"`
- `LOG_FILE = "/blue/jasondeanarnold/SPARCP/logs/audit.log"`

## Shared external dependencies

- Model/training: `torch`, `transformers`, `bitsandbytes`, `peft`, `trl`
- Data/validation: `datasets`, `pydantic`, `json`
- Retrieval: `langchain`, `langchain-chroma`, `sentence-transformers`
- Sanitization: `presidio-analyzer`, `presidio-anonymizer`
- Runtime API: `fastapi`, `uvicorn`, `langgraph`, `nemoguardrails`

- Speech: `riva-python-clients`
- 

## Notebook 1 API: Agent Training

Source: `1_SPARC_Agent_Training.md`

Data sanitization and extraction

`sanitize_text_with_presidio(text: str) -> str`

Analyzes and anonymizes text using Presidio entity replacement.

`extract_text_from_document(doc_path)`

Extracts raw text from a document via PyMuPDF (`fitz`).

RAG ingestion and vectorization

`build_vector_store(doc_paths: List[str], collection_name: str)`

Sanitizes, chunks, embeds, and persists documents to ChromaDB.

`ingest_documents(source_path: str, collection_name: str)`

Ingestion utility that converts source text, chunks, embeds, and persists to Chroma collection.

Synthetic data and schema conversion

`generate_synthetic_qa(document_chunk: str, num_pairs: int = 5)`

Generates (mocked) synthetic QA pairs and formats them in chat-message shape.

`format_to_chat_schema(raw_data: List[Dict]) -> Dataset`

Converts raw `input/output` items into HF dataset entries with `messages` list.

`load_and_process_data(agent_type: str) -> Dataset`

Loads agent-specific mock data and returns standardized chat dataset.

Fine-tuning

`run_qlora_training(train_file_path: str, output_dir: str)`

Configures quantized model loading, LoRA, training arguments, and `SFTTrainer`.

Uses:

- `BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type="nf4", ...)`
- `LoraConfig(r=16, lora_alpha=32, lora_dropout=0.05, ...)`

- `TrainingArguments(...)`
- `SFTTrainer(...)`

Validation models and schema checks

```
class CaregiverOutput(BaseModel)
```

Fields:

- `text: str`
- `emotion: str`
- `gesture: str`

```
class CoachOutput(BaseModel)
```

Fields:

- `grade: str`
- `feedback_points: List[str]`

```
class SupervisorOutput(BaseModel)
```

Fields:

- `recipient: Optional[str] = None`
- `payload: Optional[str] = None`

```
validate_agent(agent_name: str, test_prompts: List[str], model_schema: BaseModel = None)
```

Runs adapter output checks and validates mock JSON responses against Pydantic schemas.

Gradio interfaces

```
load_agent_adapter(agent_name)
```

Adapter-loader placeholder for selected agent.

```
chat_individual(message, history, agent_selection)
```

Returns simulated single-agent response based on selected persona.

```
multi_agent_orchestrator(user_message, history)
```

Simulates supervisor routing and worker-agent output assembly in one conversational trace.

Job script generation

```
generate_slurm_script()
```

Writes `train_agent.slurm` with Apptainer execution command for QLoRA training.

## Notebook 2 API: Containerization & Deployment

Source: [2\\_SPARC\\_Containerization\\_and\\_Deployment.md](#)

Container build scaffolding

`create_dockerfile()`

Writes `Dockerfile.mas` using multi-stage pattern:

- Builder stage with Poetry dependency install
- Runtime stage with slim Python image and copied artifacts

Deployment script generation

`generate_production_script()`

Writes `sparc_production.slurm` that launches:

- Riva service container
- WebSocket bridge container
- MAS container

Includes GPU and memory scheduler directives and `wait` for persistent service uptime.

Reference command block (non-function API cell)

`podman_commands` string contains local pod workflow examples for:

- Pod creation
- Riva startup
- bridge startup
- MAS startup

---

## Notebook 3 API: RIVA Backend

Source: [3\\_SPARC\\_RIVA\\_Backend.md](#)

RIVA setup helpers

`configure_riva()`

Prints Riva config expectations (ASR/TTS toggles and setup instructions).

`test_asr_service(audio_file_path)`

ASR test scaffold using `riva.client` auth/session model.

`test_tts_service(text_input)`

TTS test scaffold using `riva.client` auth/session model.

## Guardrails setup

```
create_rails_config()
```

Generates:

- `config.yml`
- `topical_rails.co`

Defines topical refusal rails for off-domain conversation boundaries.

## Multi-agent orchestration classes/functions

```
class SupervisorAgent
```

Method:

- `async process_input(self, text: str)`

```
class CaregiverAgent
```

Method:

- `async generate_response(self, text: str)`

```
class CoachAgent
```

Method:

- `async evaluate_turn(self, text: str)`

```
async handle_user_turn(audio_stream, supervisor, caregiver, coach)
```

Pipeline pattern:

1. Transcription (mocked)
2. Supervisor safety check
3. Parallel caregiver + coach tasks (`asyncio.gather`)
4. Final aggregated response

## FastAPI models and endpoints

```
class ChatRequest(BaseModel)
```

Fields:

- `session_id: str`
- `user_transcript: str`

```
class ChatResponse(BaseModel)
```

Fields:

- `caregiver_text: str`
- `caregiver_audio_b64: str`
- `caregiver_animation_cues: dict`
- `coach_feedback: str`

`GET /health`

Handler:

- `async def health_check()`

Returns service status payload.

`POST /v1/chat`

Handler:

- `async def chat_endpoint(request: ChatRequest)`

Behavior:

- Appends audit log record for request
- Invokes graph/orchestrator (`app_graph.invoke(...)`)
- Returns typed `ChatResponse`

Service launch script generation

`generate_launch_script()`

Writes `launch_backend.slurm` for persistent backend launch via `srun apptainer run --nv sparcp_backend.sif`.

---

## Endpoint Contracts

`GET /health`

### Response

```
{  
  "status": "ok",  
  "service": "SPARC-P Backend"  
}
```

`POST /v1/chat`

### Request (`ChatRequest`)

```
{  
    "session_id": "string",  
    "user_transcript": "string"  
}
```

## Response ([ChatResponse](#))

```
{  
    "caregiver_text": "string",  
    "caregiver_audio_b64": "string",  
    "caregiver_animation_cues": {},  
    "coach_feedback": "string"  
}
```

---

## Generated Artifacts

Notebook APIs create the following deployable/config files:

- [Dockerfile.mas](#)
  - [train\\_agent.slurm](#)
  - [sparc\\_production.slurm](#)
  - [launch\\_backend.slurm](#)
  - [config.yml](#)
  - [topical\\_rails.co](#)
- 

## Operational Notes

- Storage and audit paths assume HiPerGator [/blue](#) tier.
- Notebook code includes mocked sections (e.g., synthetic generation, stubbed inference) intended to be replaced with real model and service calls.
- Training and serving commands assume AppTainer-first deployment on HPC.
- API/security intent emphasizes transient PHI handling and non-sensitive audit logging.