**🧩 Epic: ATROP Phase 0 – Inception & Planning**

**📘 Description**

This Epic establishes the foundational groundwork for the ATROP protocol development. It includes defining the MVP scope, setting up the Agile workflow, structuring the GitHub repository, initializing the CI pipeline, drafting initial documentation, and aligning the Subject Matter Experts (SMEs) with development tracks. This Epic ensures that all teams are aligned, organized, and ready to enter feature-driven implementation phases with clarity and coordination.

**🎯 Objectives**

* Define the Minimum Viable Protocol (MVP) for ATROP covering FSM, AI/ML, IPC, and test scope.
* Build an actionable product backlog that maps to all major components.
* Finalize Agile methodology and cadence (Scrum/Kanban hybrid).
* Set up GitHub infrastructure: repo layout, issue templates, CI stub, and governance files.
* Assign SMEs for control plane, data plane, testing, SDKs, and documentation tracks.
* Document project architecture skeleton and community contribution guidelines.

**📦 Epic Deliverables**

* ✅ **MVP Scope Documented**: FSMs, AI engine, ML inference, IPC, and baseline test cases identified.
* ✅ **Product Backlog Drafted**: Stories broken down across modules: daemon, models, SDK, test, docs.
* ✅ **Agile Workflow Defined**: Sprint cadence, grooming process, stand-up rhythm agreed upon.
* ✅ **GitHub Repository Bootstrapped**: Directory structure, CI/CD stub, and issue/project boards created.
* ✅ **Governance Files Finalized**: LICENSE, CODE\_OF\_CONDUCT.md, CONTRIBUTING.md, SECURITY.md.
* ✅ **Initial Documentation Staged**: docs/ tree populated with intro stubs for architecture and standards.
* ✅ **Testing Strategy Defined**: Layers of testing (unit, FSM sim, integration, telemetry) outlined.
* ✅ **SMEs Assigned**: Core owners identified for each domain (Protocol, AI, ML, Test, SDK, Docs).

**🧾 User Story: T0-01 – Define MVP Scope**

**Story Points:** **5**

**Status:** ✅ **Done**

**As a** Protocol Architect, **I want to** identify and define the core ATROP components needed for the first minimum viable version. **So that** the team can prioritize development around a stable, testable baseline with clear scope and deliverables.

**🎯 Acceptance Criteria**

* Core protocol components are clearly defined for inclusion in the MVP:
  + ATROP Finite State Machine (FSM)
  + AI-based control plane engine
  + ML-based data plane inference logic
  + Inter-process communication (IPC) between components
  + Basic testing harness and simulation tooling
* MVP scope is documented in a shared location (docs/, roadmap, or shared tracker)
* MVP scope is communicated to all technical SMEs and team leads
* Scope enables dependencies for T0-02 (backlog drafting) and T0-07 (test strategy)
* The story is reviewed and signed off by product leadership

**🧩 Notes**

* Aligns with Phase 0 Epic: *Inception & Planning*
* Based on design documents and strategic roadmap
* Output feeds directly into backlog generation and SME assignments

**🧾 User Story: T0-02 – Draft Product Backlog**

**Story Points:** **8**  
**Status:** 🟡 **In Progress**

**As a** Product Owner and Lead Developer  
**I want to** create a complete Epic and Story breakdown for all core ATROP modules  
**So that** the development team has a clear, actionable, and prioritized backlog aligned with the project vision and MVP scope.

**🎯 Acceptance Criteria**

* Epics and user stories are created for:
  + Control Plane (FSM, AI engine, intent processing)
  + Data Plane (ML inference, FIF generation, trust scoring)
  + Inter-process Communication (IPC protocols and daemons)
  + SDKs (gRPC, Python, C++, telemetry interfaces)
  + Testing (unit, FSM simulation, integration, CI hooks)
  + Docs and standards alignment
* Each story includes description, estimate (points), SME, and dependencies
* Stories are structured under related Epics in a tracking system (GitHub Projects, Jira, etc.)
* Backlog aligns with MVP scope defined in T0-01
* Reviewed by tech leads and approved for Phase 1 execution

**🧩 Notes**

* Enables sprint planning and milestone tracking for Phases 1 through 7
* Backlog stories will be imported into project management tooling
* This task blocks further breakdown of sprint scopes and role assignment

**🧾 User Story: T0-02 – Draft Product Backlog**

**Story Points:** **8**  
**Status:** 🟡 **In Progress**  
**Predecessor:** T0-01 – Define MVP Scope  
**Predecessor Status:** ✅ **Done**

**As a** Product Owner and Lead Developer, **I want to** create a complete Epic and Story breakdown for all core ATROP modules, **So that** the development team has a clear, actionable, and prioritized backlog aligned with the project vision and MVP scope.

**🎯 Acceptance Criteria**

* Epics and user stories are created for:
  + Control Plane (FSM, AI engine, intent processing)
  + Data Plane (ML inference, FIF generation, trust scoring)
  + Inter-process Communication (IPC protocols and daemons)
  + SDKs (gRPC, Python, C++, telemetry interfaces)
  + Testing (unit, FSM simulation, integration, CI hooks)
  + Documentation & standards alignment
* Each story includes:
  + Description
  + Story Points
  + SME assignment
  + Dependencies (if applicable)
  + Status tracker
* All stories are organized under Epics using a shared tracking system (GitHub Projects, Jira, Excel tracker)
* Backlog is validated to fully cover the MVP scope output from T0-01
* Reviewed and confirmed by protocol architect, test lead, and SME owners for readiness

**🧩 Notes**

* T0-01 output is a required input to ensure accurate scope coverage
* This user story is a blocker for roadmap alignment (T0-08) and SME assignment (T0-09)
* Results in a living backlog for Phases 1–7, enabling sprint-based execution across teams

**🧾 User Story: T0-03 – Select Agile Workflow**

**Story Points:** **2**  
**Status:** ✅ **Done**  
**Predecessor:** None

**As a** Scrum Master, **I want to** choose and define the Agile methodology, sprint cadence, and team rituals. **So that** the ATROP team has a consistent and effective process to manage work execution across all phases.

**🎯 Acceptance Criteria**

* Chosen methodology is defined (Scrum, Kanban, or hybrid)
* Sprint cadence is documented (e.g., 2-week sprints)
* Regular Agile ceremonies are scheduled:
  + Daily stand-ups
  + Backlog grooming
  + Sprint planning
  + Sprint review & retrospectives
* Workflow is communicated to all contributors and reflected in GitHub Project Board (or Jira)
* Initial burndown and story tracking templates are shared

**🧩 Notes**

* Establishes rhythm for task execution starting in Phase 1
* Aligns with T0-02 (backlog structure) and T0-05 (project board automation)
* Enables team-wide consistency for hybrid remote/in-person collaboration

**🧾 User Story: T0-04 – Setup GitHub Repo Structure**

**Story Points:** **3**  
**Status:** ✅ **Done**  
**Predecessor:** None

**As a** DevOps Engineer, **I want to** initialize the GitHub repository with the required directory structure and placeholder files. **So that** each development stream (control plane, data plane, SDK, test, docs) has a standardized starting point and clear separation of concerns.

**🎯 Acceptance Criteria**

* Repository includes top-level folders:
  + daemon/ (with control\_plane/, data\_plane/, ipc/)
  + test/ (unit, integration, fsm-simulations)
  + sdk/ (gRPC, Python, C++, telemetry)
  + models/ (training, evaluation, pretrained, export)
  + docs/, examples/, specs/, tools/
* Each directory contains a README.md stub describing its purpose
* .gitignore, .editorconfig, and any common .vscode/ config are committed
* Repository is pushed and accessible in the ATROP GitHub organization
* Directory layout aligns with ATROP’s software architecture from the design index

**🧩 Notes**

* Enables future CI/CD integration (T0-11) and doc generation (T0-10)
* Forms the foundation for Phase 1 workstreams
* Verified against structure outlined in Atrop-SWP.txt and whitepaper specs

**🧾 User Story: T0-05 – Create GitHub Project Board**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T0-04 – Setup GitHub Repo Structure  
**Predecessor Status:** ✅ **Done**

**As a** Scrum Master, **I want to** set up the GitHub Project Board with structured columns and templates. **So that** the team can track progress clearly, manage workflow transitions, and automate story state handling across sprints.

**🎯 Acceptance Criteria**

* GitHub Project Board is created under the ATROP repository or organization
* Columns are added: Backlog, To Do, In Progress, Review, Done
* Issue templates are created under .github/ISSUE\_TEMPLATE/:
  + bug\_report.md
  + story\_task.md or equivalent
* Project board automation rules are configured (e.g., auto-move issues on status update or label change)
* Labels for status, priority, and module type are defined and documented
* The board is reviewed and shared with all contributors as the central workflow tool

**🧩 Notes**

* Enables agile delivery tracking and visual progress for all Phases
* Dependent on T0-04 repository being initialized and visible
* Will integrate with CI triggers in T0-11 and be used for sprint reviews starting Phase 1

**🧾 User Story: T0-06 – Review Governance Files**

**Story Points:** **2**  
**Status:** ✅ **Done**  
**Predecessor:** T0-04 – Setup GitHub Repo Structure  
**Predecessor Status:** ✅ **Done**

**As a** Legal Advisor and Community Manager, **I want to** finalize the core governance and compliance files for the ATROP project. **So that** contributor expectations, legal clarity, and community safety are clearly defined and enforced from day one.

**🎯 Acceptance Criteria**

* LICENSE is selected and finalized (ensures IP and attribution are covered per ATROP copyright terms)
* CODE\_OF\_CONDUCT.md defines expected behavior, enforcement, and reporting contacts
* CONTRIBUTING.md provides contribution workflow, pull request etiquette, and code review process
* SECURITY.md outlines vulnerability disclosure policy and contact instructions
* All files are committed to the root of the GitHub repo and rendered correctly
* Governance files are reviewed by project leadership and linked in the README.md

**🧩 Notes**

* Establishes compliance with open-source norms and vendor-alignment requirements
* Supports GitHub’s community health checklist for visibility and credibility
* Enables safe onboarding for new contributors in Phases 1–N

**🧾 User Story: T0-07 – Define Test Strategy (MVP)**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T0-01 – Define MVP Scope  
**Predecessor Status:** ✅ **Done**

**As a** Test Architect  
**I want to** define a comprehensive test strategy for the ATROP MVP  
**So that** all protocol layers and components have clear, validated testing plans from the beginning of implementation.

**🎯 Acceptance Criteria**

* Testing layers are clearly defined:
  + Unit tests (protocol logic, model I/O, FSM transitions)
  + FSM simulation (state transitions, correction flows, zone behavior)
  + Integration tests (control ↔ data plane, IPC messaging, AI feedback)
  + Telemetry validation (FIF creation, export, and interpretation)
* Test types are mapped to the modules in the MVP scope (defined in T0-01)
* Toolchain is selected and documented (pytest, gtest, custom FSM sim engine, etc.)
* Test coverage goals are drafted (e.g., 80%+ for core logic)
* Deliverables include a strategy document under docs/ and CI integration considerations
* Validated by protocol and AI/ML leads for accuracy and alignment

**🧩 Notes**

* Affects CI design in T0-11 and drives test harness creation in Phases 1, 6, and 7
* Enables early test-driven development and fault isolation in a modular design approach
* Output document should guide implementation starting from Phase 1 onward

**🧾 User Story: T0-08 – Align Roadmap Milestones**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T0-02 – Draft Product Backlog  
**Predecessor Status:** 🟡 **In Progress**

**As a** Project Manager, **I want to** lock the sprint-level roadmap aligned with the ATROP draft design index. **So that** the team has a shared execution timeline that reflects priority modules like AI/ML engine, protocol FSM, and vendor adapter workstreams.

**🎯 Acceptance Criteria**

* Sprint roadmap is defined for all major components:
  + Control Plane (FSM, AI model engine)
  + Data Plane (ML inference, trust scores, FIF)
  + IPC layer (messaging, lifecycle)
  + SDKs and telemetry interfaces
  + Testing tracks (unit, integration, simulation)
  + Vendor adapters and platform compatibility
* Roadmap includes:
  + Sprint themes and scope
  + Cross-phase dependencies
  + SME ownership per track
* Visual roadmap or timeline chart (Gantt or table) is stored in docs/
* Reviewed with product owner, tech leads, and Scrum Master
* Shared with all contributors via GitHub or project workspace

**🧩 Notes**

* Cannot complete until T0-02 backlog is finalized
* Sets timing boundaries for Phase 1 through Phase 8 execution
* Roadmap serves as a reference baseline for velocity tracking and release milestones

**🧾 User Story: T0-09 – Assign SMEs to Tracks**

**Story Points:** **2**  
**Status:** 🔵 **To Do**  
**Predecessor:** T0-02 – Draft Product Backlog  
**Predecessor Status:** 🟡 **In Progress**

**As a** Tech Lead, **I want to** assign Subject Matter Experts (SMEs) to each major functional track of the ATROP project. **So that** responsibilities are clearly distributed, and development ownership is aligned with technical expertise.

**🎯 Acceptance Criteria**

* SME roles are identified and assigned for the following core areas:
  + Protocol logic & FSM
  + AI control plane (GNN, policy enforcement)
  + ML data plane (inference, telemetry)
  + Platform integration (eBPF, Netlink, systemd)
  + SDK/API layer (Python, C++, gRPC, telemetry)
  + Testing & CI
  + Documentation & standards alignment
* SME assignments are documented in a shared resource (docs/team\_roles.md or equivalent)
* Each SME confirms their track assignment and understands expected scope
* SME assignments are referenced in the project board and story assignments
* Reviewed with project manager and product owner

**🧩 Notes**

* SME ownership ensures depth and continuity across sprints
* Enables precise task routing, clearer estimations, and technical accountability
* Must align with the backlog hierarchy established in T0-02

**🧾 User Story: T0-10 – Create Initial Dev Docs**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T0-01 – Define MVP Scope → ✅ **Done**
* T0-06 – Review Governance Files → ✅ **Done**

**As a** Documentation SME, **I want to** create initial development documentation in the docs/ directory. **So that** contributors and stakeholders have a clear reference for ATROP’s architecture, standards, and design scope from the start.

**🎯 Acceptance Criteria**

* docs/ directory includes the following intro stubs:
  + architecture/README.md – Overview of control/data plane, FSMs, AI/ML layers
  + diagrams/README.md – Placeholder with structure for FSMs, data paths, learning flows
  + whitepaper/README.md – Link or reference to ATROP whitepaper draft
  + standards/README.md – Alignment notes with IETF, IEEE, and vendor platform compliance
* Files are correctly formatted in markdown and committed to the repo
* Content reflects the MVP scope (T0-01) and governance guidelines (T0-06)
* Reviewed by protocol architect and product owner
* Ready for extension in later documentation sprints (e.g., specs, tutorials, deployment)

**🧩 Notes**

* Documentation is critical for onboarding contributors and vendor reviewers
* Aligns with open-source best practices and GitHub community standards
* Enables early feedback loop for architecture assumptions and standards compliance

**🧾 User Story: T0-11 – CI Pipeline Bootstrap**

**Story Points:** **3**  
**Status:** 🟡 **In Progress**  
**Predecessor:** T0-04 – Setup GitHub Repo Structure  
**Predecessor Status:** ✅ **Done**

**As a** DevOps Engineer, **I want to** set up a baseline CI pipeline in .github/workflows/ci.yml. **So that** all code contributions are automatically linted and validated through placeholder test stages, enabling safe, consistent commits from the start.

**🎯 Acceptance Criteria**

* CI pipeline is defined using GitHub Actions and committed to .github/workflows/ci.yml
* Linting step runs on supported languages (Python, C++, YAML/Markdown)
* Placeholder test step runs a minimal test script (e.g., dummy pytest, gtest, or shell test)
* Workflow is triggered on push, pull\_request, and main branch protection
* Logs are accessible in GitHub Actions UI with pass/fail summaries
* Pipeline is documented briefly in CONTRIBUTING.md

**🧩 Notes**

* This bootstrap is foundational for full test automation in Phases 6 and 7
* Will be extended later with per-module tests and coverage reporting
* Enables merge gatekeeping and PR validation starting Phase 1

**🧩 Epic: ATROP Phase 1 – Base Infrastructure Setup**

**📘 Description**

This Epic establishes the foundational software and development infrastructure for the ATROP protocol stack. It covers everything from scaffolding the build system and initializing code structure to configuring the local dev environment, CI pipeline, and test runners. The goal is to enable clean, parallel development across all protocol components from Phase 2 onward.

**🎯 Objectives**

* Scaffold the multi-language build system to support modular ATROP compilation and test runs
* Initialize clean entrypoints and folder layout across daemon/, sdk/, test/, and related modules
* Set up a repeatable and containerized development environment for consistent tooling across teams
* Bootstrap initial test scaffolds and CI integration for automated validation
* Provide shared services for configuration loading and logging abstraction
* Prepare protocol message parsing stubs as placeholders for FSM-driven logic

**📦 Epic Deliverables**

* ✅ Makefile, setup.py, or CMake scaffolding for core modules
* ✅ Entry stubs created for control plane, data plane, and IPC daemons
* ✅ Fully structured GitHub repo with placeholders or .keep in all relevant folders
* ✅ pytest and gtest harnesses set up with dummy test runners
* ✅ Dockerfile and .devcontainer/ configured for standardized local setup
* ✅ Extended CI pipeline (ci.yml) with linting and test stages
* ✅ Shared config loader (YAML/JSON parser) usable across modules
* ✅ Logger abstraction supporting CLI, file, and structured outputs
* ✅ Stub protocol handlers for Discovery, Decision, and Observation messages
* ✅ A single dummy test in each module to ensure end-to-end pipeline validation

**🧾 User Story: T1-01 – Scaffold Build System**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** Phase 0 Completion

**As a** DevOps Engineer, **I want to** scaffold the build system for the ATROP codebase. **So that** developers can reliably build and run components in daemon/, sdk/, and test/ directories across languages and modules.

**🎯 Acceptance Criteria**

* A Makefile, setup.py, or CMakeLists.txt is created based on language needs (C++, Python)
* Targets or commands are available to:
  + Build daemon/ (control plane, data plane, IPC)
  + Build sdk/ (Python, C++, gRPC stubs if available)
  + Run or install test/ dependencies
* Build outputs are directed to build/ or relevant temp/output folders
* All commands work on Linux-based environments and support future container builds
* Build instructions are added to README.md or docs/dev\_setup.md

**🧩 Notes**

* Enables T1-02 (Entrypoints), T1-04 (Testing Scaffold), and T1-05 (Dev Environment)
* Modular support for mixed-language build targets is expected
* Initial tooling should be minimal, extensible, and cross-platform compatible

**🧾 User Story: T1-02 – Initialize Module Entrypoints**

**Story Points:** **2**  
**Status:** 🔵 **To Do**  
**Predecessor:** Phase 0 Completion

**As a** Core Developer, **I want to** add initial entrypoint files (main.cpp, main.py) in key daemon modules. **So that** each major component (control plane, data plane, IPC) has a bootable structure that can evolve with real logic during implementation phases.

**🎯 Acceptance Criteria**

* main.cpp is created under daemon/control\_plane/ and daemon/ipc/ with minimal compile-ready structure
* main.py is created under daemon/data\_plane/ with a basic if \_\_name\_\_ == "\_\_main\_\_" block
* Each stub file includes:
  + Module description comment
  + Placeholder for argument parsing or future service registration
  + Return exit code 0 on success
* Build system (from T1-01) successfully compiles or runs each entrypoint
* Entrypoints are documented in each module’s README.md for reference

**🧩 Notes**

* Enables T1-03 (Folder structure), T1-07 (Config Loader), and T1-09 (Protocol Handlers)
* Control and data planes may use different runtimes (C++ for performance, Python for ML flexibility)
* Provides a clean and testable scaffold for CI smoke test in T1-10

**🧾 User Story: T1-03 – Finalize Directory Structure**

**Story Points:** **1**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-02 – Initialize Module Entrypoints  
**Predecessor Status:** 🔵 **To Do**

**As a** Repo Maintainer, **I want to** create placeholder folders and basic metadata files in all major code areas. **So that** the repository is fully structured and ready for modular development, navigation, and documentation.

**🎯 Acceptance Criteria**

* Directories created for all key modules if not already present:
  + daemon/control\_plane/, daemon/data\_plane/, daemon/ipc/
  + test/unit/, test/integration/, test/fsm-simulations/
  + sdk/python/, sdk/c++/, sdk/grpc/, sdk/telemetry/
  + models/pretrained/, models/training\_data/, models/evaluation/
  + docs/architecture/, docs/standards/, docs/diagrams/, docs/whitepaper/
* Each directory contains a .keep or README.md file with a short placeholder description
* No folder is left empty to avoid pruning during version control commits
* All changes are pushed to the main repository with a commit titled Finalize repo structure with placeholders
* Repo structure matches that defined in the Atrop-SWP.txt and whitepaper documentation

**🧩 Notes**

* Enables discoverability and onboarding for all future contributors
* Helps with project board organization and component-based planning
* Required before documentation, CI paths, or module imports can be reliably managed

**🧾 User Story: T1-04 – Setup Testing Scaffold**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-01 – Scaffold Build System  
**Predecessor Status:** 🔵 **To Do**

**As a** Test Engineer, **I want to** set up the initial testing framework and folder structure for Python and C++ code. **So that** developers can write and run unit tests consistently across modules using standardized tools.

**🎯 Acceptance Criteria**

* pytest is initialized with test/unit/ structure for Python modules (e.g., data\_plane, sdk)
* gtest is initialized with CMake or Makefile targets under test/unit/ for C++ modules (e.g., control\_plane, ipc)
* Folder structure includes:
  + test/unit/
  + test/integration/
  + test/fsm-simulations/
* Sample/dummy test files are included in each testing directory to validate runner setup
* Testing commands are runnable via CLI and CI (e.g., make test, pytest, ctest)
* Documented in README.md under test/ directory

**🧩 Notes**

* Unblocks T1-10 (Dummy Unit Test) and Phase 6 testing efforts
* Required for CI test trigger extension in T1-06
* Helps enforce test-first development practices from Phase 2 onward

**🧾 User Story: T1-05 – Create Local Dev Environment**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-01 – Scaffold Build System  
**Predecessor Status:** ✅ **Done**

**As a** DevOps Engineer, **I want to** create a containerized local development environment using Docker and .devcontainer/. **So that** contributors can develop, build, and test ATROP components in a consistent, reproducible environment across platforms.

**🎯 Acceptance Criteria**

* A Dockerfile is created using a minimal base image (e.g., Ubuntu or Debian)
* .devcontainer/devcontainer.json is configured to support VS Code remote containers (optional but recommended)
* Environment includes pre-installed dependencies:
  + Python 3.x, pip, and pytest
  + C++ toolchain (g++, cmake, gtest)
  + Git, curl, make, and debugging tools
* Shared developer configuration included:
  + Common shell aliases in .bashrc or .zshrc
  + Mounted volumes for daemon/, sdk/, test/, etc.
* Environment must build all modules successfully using T1-01 scaffolding
* Documented usage instructions placed in docs/dev\_setup.md or README.md

**🧩 Notes**

* Enables all team members to develop on a unified toolchain with minimal local setup
* Required to validate downstream tasks like T1-06 (CI extension) and T1-10 (unit test validation)
* Ensures clean handoff between individual workstations and CI/CD environments

**🧾 User Story: T1-06 – Extend CI Pipeline**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ **Done**

**As a** DevOps Engineer, **I want to** extend the existing GitHub Actions CI pipeline to include test and lint stages for Python and C++. **So that** all contributions are automatically validated for code quality and functional correctness across all supported languages.

**🎯 Acceptance Criteria**

* .github/workflows/ci.yml is updated with jobs for:
  + Python linting (e.g., flake8, black, or ruff)
  + Python testing (pytest) under test/unit/
  + C++ linting (e.g., clang-tidy, cpplint)
  + C++ testing (gtest) with build commands from T1-01
* CI runs on push and pull\_request triggers
* Logs display lint and test results with clear pass/fail status
* Failing steps block merges via branch protection rules
* CI pipeline changes are reviewed and validated on at least one PR
* CI instructions are documented in CONTRIBUTING.md or README.md

**🧩 Notes**

* Builds directly on T0-11 and extends coverage to all runnable test artifacts
* Enables verification for early dummy tests (T1-10) and unlocks real test integration in Phase 2 onward
* Aligns with GitHub Projects automation and contributor workflow compliance

**🧾 User Story: T1-07 – Central Config Loader**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-02 – Initialize Module Entrypoints  
**Predecessor Status:** ✅ **Done**

**As an** Infrastructure Engineer, **I want to** implement a centralized configuration loader that supports JSON and YAML formats. **So that** all ATROP daemon modules can share a consistent, flexible config structure for runtime parameters.

**🎯 Acceptance Criteria**

* A config loader module is created (e.g., config\_loader.py, config\_loader.cpp)
* Supports both .json and .yaml formats using appropriate parsers:
  + Python: json, PyYAML
  + C++: nlohmann/json, yaml-cpp
* Configuration fields include:
  + Module parameters (e.g., ports, timeouts, logging levels)
  + Environment flags (e.g., dev/staging mode, test toggles)
  + Paths (e.g., model directory, data files)
* Config loader is imported and used in all main.\* files under daemon/
* Fails gracefully on parse errors with descriptive logs
* Includes unit tests and validation with sample config.yaml and config.json files
* Documented usage in docs/dev\_setup.md and per-module README.md

**🧩 Notes**

* Enables dynamic configuration for downstream modules including FSM logic, AI model loading, and telemetry paths
* Serves as a baseline for runtime reload support and future config-driven behavior (e.g., in Phase 5 and 10)
* Blocks T1-08 (Logging Abstraction) and must be compatible with containerized environments (T1-05)

**🧾 User Story: T1-08 – Logging Abstraction**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-07 – Central Config Loader  
**Predecessor Status:** ✅ **Done**

**As an** Infrastructure Engineer, **I want to** implement a unified logging abstraction for both daemon modules and test infrastructure. **So that** all ATROP components can emit consistent, structured, and configurable logs for debugging, telemetry, and observability.

**🎯 Acceptance Criteria**

* A logging utility is created in both Python and C++:
  + logger.py using logging and/or jsonlogger
  + logger.cpp using a minimal wrapper around spdlog or standard streams
* Logging supports:
  + Console logging with human-readable output
  + File logging with rotating file or flat log options
  + JSON log format for machine parsing
* Log level and format are configurable via the central config loader (T1-07)
* Supported log levels: DEBUG, INFO, WARNING, ERROR, CRITICAL
* Integrated into all main.\* daemon entrypoints and basic test files
* Includes unit tests for:
  + Format validation
  + Config-driven behavior
  + Multi-target output (console + file)
* Sample output is shown in docs/dev\_setup.md or module README.md

**🧩 Notes**

* Enables structured observability and log parsing during simulations and integration tests (Phase 6–7)
* Will later feed into telemetry and feedback modules during AI/ML decision tracing
* Must work cleanly in both local environments (T1-05) and inside CI containers

**🧾 User Story: T1-09 – Add Placeholder Protocol Handlers**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-02 – Initialize Module Entrypoints  
**Predecessor Status:** ✅ **Done**

**As a** Protocol Engineer, **I want to** create empty/stub message handlers for key ATROP protocol packet types. **So that** the protocol stack has defined placeholders for parsing and dispatch logic for core message families.

**🎯 Acceptance Criteria**

* Stub handler functions or classes are created for:
  + Discovery packets (initial node intro, topology discovery)
  + Decision packets (AI-driven routing decisions or policies)
  + Observation packets (telemetry, trust updates, anomaly feedback)
* Handlers exist in appropriate locations:
  + daemon/control\_plane/
  + daemon/ipc/
* Each stub includes:
  + Function signature and message type identifier
  + Basic parsing hook or switch case
  + TODO/FIXME comments with reference to expected structure (e.g., NIV, PIV, IDR, FIF)
* Compilation and syntax check passes via CI
* Each handler is linked to a logging output via T1-08 abstraction
* Documented in module README.md with a note that full parsing will follow in Phase 2

**🧩 Notes**

* Required for T2-03 (Create Packet Parser & Dispatcher)
* Enables initial simulation of control/data packet flow through the FSM
* Forms a clean stub interface for integration with AI/ML decision and telemetry paths later in the pipeline

**🧾 User Story: T1-10 – Add Dummy Unit Test**

**Story Points:** **1**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ **Done**

**As a** Test Engineer, **I want to** add one minimal passing unit test in each module, **So that** the test infrastructure and CI pipeline are validated as functional and ready for real test development.

**🎯 Acceptance Criteria**

* A dummy test is added for each key module:
  + daemon/control\_plane/ (C++ with gtest)
  + daemon/data\_plane/ (Python with pytest)
  + daemon/ipc/ (C++ or Python)
* Each test must:
  + Compile or run without errors
  + Pass successfully
  + Assert a trivial condition (e.g., EXPECT\_EQ(1, 1) or assert True)
* Tests are executed automatically via the CI pipeline (T1-06)
* Test results are visible in GitHub Actions CI logs
* Dummy test files are marked with comments indicating purpose and next steps
* Documented in test/README.md as a CI smoke test pattern

**🧩 Notes**

* This validates the full path: local test → commit → CI trigger → pass
* Confirms that the testing scaffold and CI jobs are properly wired
* Provides a baseline for test-driven development in Phase 2 and beyond

**🧩 Epic: ATROP Phase 2 – Protocol State Machine & Parsing Core**

**📘 Description**

This Epic focuses on building the foundational protocol state logic for ATROP’s control plane. It covers finite state machine (FSM) implementation, state transitions, packet handling, and protocol header parsing. It also introduces Autonomous Topology Zones (ATZ) and integrates initial policy-awareness mechanisms. This phase forms the cognitive engine of ATROP, turning passive flow data and intent into active, state-driven behavior.

**🎯 Objectives**

* Implement the complete ATROP FSM core with 9 primary states
* Encode all state transitions and events to drive autonomous routing behavior
* Develop the packet parsing engine and dispatcher for Discovery, Decision, Correction, and other messages
* Integrate FSM logic with topology-aware state transitions using ATZ formation
* Add policy validation checkpoints to enforce intent-driven behavior
* Build robust test and simulation coverage for FSM behavior and state integrity

**📦 Epic Deliverables**

* ✅ FSM implemented with all ATROP core states:
  + INIT
  + DISCOVERY
  + LEARN
  + DECIDE
  + ENFORCE
  + OBSERVE
  + FEEDBACK
  + CORRECT
  + EXIT
* ✅ State transition engine with event-driven architecture
* ✅ Protocol packet dispatcher with stubbed message families
* ✅ Header parsers for NIV, PIV, IDR, FIF (binary or TLV format)
* ✅ ATZ zone logic: formation, identity, and zone-based FSM behavior
* ✅ Hooks for intent/policy evaluation before state transitions
* ✅ FSM fallback on correction feedback and anomaly events
* ✅ Unit tests for FSM transitions and edge cases
* ✅ ATZ simulation for join/split behavior
* ✅ Protocol flow documented with sequence diagrams

**🧾 User Story: T2-01 – Define Core ATROP States**

**Story Points:** **8**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-02 – Initialize Module Entrypoints  
**Predecessor Status:** ✅ **Done**

**As a** Protocol Architect  
**I want to** define and implement the full set of core states in the ATROP finite state machine (FSM)  
**So that** the protocol can execute its autonomous routing logic through deterministic, reactive, and policy-driven transitions.

**🎯 Acceptance Criteria**

* All 9 ATROP FSM states are implemented with structured logic and placeholders:
  + INIT: Agent bootstraps and prepares for registration
  + DISCOVERY: Neighbor and boundary discovery
  + LEARN: Ongoing or completed training based on topological input
  + DECIDE: AI inference and intent-based route selection
  + ENFORCE: Policy enforcement and traffic direction
  + OBSERVE: Continuous telemetry and behavior monitoring
  + FEEDBACK: Reaction to inference results and data thresholds
  + CORRECT: Fault handling and fallback upon anomalies
  + EXIT: Clean shutdown or security-triggered isolation
* Each state is defined as a class or function with:
  + Input triggers
  + Valid transitions
  + Logging using T1-08 logger
  + Integration with config from T1-07
* Transitions are encoded according to event flow (based on FSM diagram)
* States respond to anomaly/SLA violations, AI confidence thresholds, and feedback loops
* Placeholder action hooks included (e.g., model updates, shutdown, telemetry start)
* Documented in docs/fsm/README.md with references to state responsibilities and transitions

**🧩 Notes**

* Establishes the base for all FSM-driven logic in Phase 2–7
* Must support visual diagram:

A diagram of a company

AI-generated content may be incorrect.

* Required for implementing transitions in T2-02 and zone-based logic in T2-05
* Ties directly into packet interpretation (T2-03), policy hooks (T2-06), and correction handling (T2-07)

**🧾 User Story: T2-02 – Implement FSM Transitions**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-01 – Define Core ATROP States  
**Predecessor Status:** ✅ **Done**

**As a** Protocol Engineer, **I want to** encode all event-driven transitions between FSM states in the ATROP state engine. **So that** the system can autonomously progress through states based on real-time events, telemetry, and policy logic.

**🎯 Acceptance Criteria**

* All valid transitions between the 9 FSM states are encoded in a central state engine/router
* Transitions respond to internal events such as:
  + Anomaly/SLA violation
  + AI decision confidence thresholds
  + Telemetry collection completion
  + Model retraining signals
  + Zone stability changes
* Transition rules match the approved FSM diagram (uploaded PNG)
* Invalid transitions are blocked and logged as warnings or errors
* Transition engine integrates:
  + Logging via T1-08
  + Config-driven thresholds from T1-07
* Unit tests simulate event triggers to verify:
  + Valid paths (e.g., LEARN → DECIDE → ENFORCE)
  + Reactive fallback (ENFORCE → CORRECT → EXIT)
  + Restart/resume paths (e.g., CORRECT → LEARN)
* Documented flowchart or table summarizing allowed state transitions is added to docs/fsm/transitions.md

**🧩 Notes**

* Foundation for dynamic FSM runtime behavior throughout all ATROP modules
* Required for policy enforcement hooks (T2-06), correction handling (T2-07), and test simulations (T2-08)
* Ensures full lifecycle flow from INIT to EXIT is deterministic, observable, and fault-tolerant

**🧾 User Story: T2-03 – Create Packet Parser & Dispatcher**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T1-09 – Add Placeholder Protocol Handlers  
**Predecessor Status:** 🔵 **To Do**

**As a** Core Developer, **I want to** implement a packet parsing and dispatching module for ATROP message families. **So that** all incoming protocol packets are correctly identified, decoded, and routed to the appropriate FSM or control handler.

**🎯 Acceptance Criteria**

* Central packet dispatcher is implemented (e.g., packet\_dispatcher.cpp/.py)
* Dispatcher recognizes and routes messages based on type:
  + Discovery
  + Decision
  + Correction
  + Observation/Telemetry
* Packet structure includes:
  + Standard ATROP headers (message ID, type, source, destination, sequence)
  + Optional payloads for future extensibility
* Parsing logic extracts message fields and validates type + size
* Dispatcher invokes stub handlers from T1-09
* All parsing errors are logged via the unified logger (T1-08)
* Parser supports both simulated (test) and real packet sources (future integration)
* Unit tests verify correct dispatch for each message type using mock inputs
* Documented in docs/protocol/dispatcher.md with message examples

**🧩 Notes**

* Enables FSM transitions to be triggered from real packet input
* Required for T2-04 (Protocol Header Parsing) and T2-07 (Correction Path)
* Initial implementation may use fixed field formats (upgradeable to TLV or protobuf later)

**🧾 User Story: T2-04 – Parse Protocol Headers**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-03 – Create Packet Parser & Dispatcher  
**Predecessor Status:** 🔵 **To Do**

**As a** Protocol Developer, **I want to** implement parsing logic for ATROP packet headers. **So that** each message can be accurately decoded and interpreted based on standard fields used in routing, telemetry, and inference feedback.

**🎯 Acceptance Criteria**

* Parser module implemented to extract and decode the following headers:
  + NIV – Node Identity Vector
  + PIV – Path Inference Vector
  + IDR – Intent/Decision Record
  + FIF – Feedback Injection Format
* Header format is defined using:
  + Fixed field binary layout **or**
  + TLV (Type-Length-Value) encoding format
* Parser performs:
  + Field validation (e.g., length, checksum if applicable)
  + Versioning support for forward compatibility
  + Integration with dispatcher output (T2-03)
* Parsed fields are passed to FSM or policy engine for contextual processing
* Logging occurs for successful and failed parses using T1-08
* Unit tests simulate parsing for each header with mock packet bytes
* Parsing format and field structure documented in docs/protocol/headers.md with diagrams

**🧩 Notes**

* Required for downstream policy logic (T2-06), AI feedback (T3-04), and telemetry injection (T4-04)
* Initial implementation may assume fixed-length fields but must leave room for TLV evolution
* May later link to Protobuf schema or YANG model in Phases 5 and 9

**🧾 User Story: T2-05 – Implement ATZ Zone Formation**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-01 – Define Core ATROP States  
**Predecessor Status:** ✅ **Done**

**As a** Topology Developer, **I want to** implement logic for Autonomous Topology Zone (ATZ) formation and zone ID assignment. **So that** ATROP nodes can organize themselves into logical FSM-bound domains for localized learning, enforcement, and decision-making.

**🎯 Acceptance Criteria**

* ATZ formation logic is implemented to:
  + Detect local topological boundaries
  + Cluster nodes into autonomous zones based on metrics (latency, role, region)
  + Assign a unique zone\_id to each ATZ instance
* Zone map is maintained as a data structure or database (e.g., in-memory map or Redis stub)
* FSM instances are scoped and managed per-zone
* Cross-zone communication rules are defined (even if deferred for Phase 10.2)
* Telemetry and learning data are tagged by zone\_id for localized control
* Includes a simulation hook to test dynamic zone formation based on node discovery
* Logging added for zone events (created, joined, split, merged)
* Documented in docs/fsm/zones.md with diagrams and zone lifecycle flow

**🧩 Notes**

* Required for T2-09 (Simulate ATZ Join/Split) and future federated ML logic
* May include config-driven behavior for zone granularity or static override
* Zone logic must be compatible with FSM state scopes and transition rules

**🧾 User Story: T2-06 – Embed Policy Awareness Hooks**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-02 – Implement FSM Transitions  
**Predecessor Status:** ✅ **Done**

**As a** Policy/Intent Developer, **I want to** embed policy-awareness hooks into the ATROP FSM state engine. **So that** transitions and routing decisions can be constrained and validated against user or service-level intents.

**🎯 Acceptance Criteria**

* Hooks are integrated into the FSM transition engine (from T2-02)
* Policy checks are triggered before key transitions (e.g., LEARN → DECIDE, DECIDE → ENFORCE)
* Intent types supported in checks (initially as stub):
  + SLA targets (latency, bandwidth)
  + Zone inclusion/exclusion
  + Security levels
  + Flow types (e.g., real-time, bulk, secure)
* Policy evaluation can:
  + Approve the transition
  + Block and revert the transition
  + Trigger a fallback or CORRECT state
* Integrates with placeholder Intent Processing Unit (T3-03) as a stub
* Violations are logged with reason and blocked transition context
* Unit tests simulate allowed/denied transitions using mock policies
* Hooks documented in docs/policy\_hooks.md with sample use cases

**🧩 Notes**

* Enables enforcement of network-wide constraints driven by external policy inputs
* Will evolve into full Intent Processing + Decision Orchestration path in Phase 3
* Also aligns with simulated policy-based routing in Phase 7 (T7-03)

**🧾 User Story: T2-07 – Add Correction Packet Handler**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T2-01 – Define Core ATROP States → ✅ Done
* T2-02 – Implement FSM Transitions → ✅ Done

**As a** Protocol Developer, **I want to** implement a handler for correction packets. **So that** ATROP nodes can respond to anomaly or SLA violations by transitioning into fallback or isolation states within the FSM.

**🎯 Acceptance Criteria**

* A correction packet handler is implemented and integrated into the dispatcher (T2-03)
* Correction packets trigger evaluation of:
  + Anomaly type (e.g., SLA breach, trust score drop)
  + Source and impact scope (zone-wide or node-specific)
* FSM reacts by triggering CORRECT → ISOLATE or CORRECT → EXIT transitions
* Handler can:
  + Log violation details
  + Throttle affected flows or trigger FSM restart
  + Signal the Feedback and LEARN state if correction is recoverable
* Configurable thresholds define when fallback is executed
* Unit tests simulate injection of correction packets and expected state reactions
* Documented in docs/protocol/correction\_handler.md with examples and state flow

**🧩 Notes**

* Closely tied to FEEDBACK and OBSERVE loop from Phase 1 and FSM logic in Phase 2
* Enables reactive safety logic and anomaly containment for trust-aware routing
* Will be extended in Phase 4 with FIF telemetry and adaptive inference rollback

**🧾 User Story: T2-08 – Build FSM Unit Test Suite**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-01 – Define Core ATROP States  
**Predecessor Status:** ✅ Done

**As a** Test Engineer, **I want to** implement a complete unit test suite for the ATROP FSM. **So that** all valid and invalid state transitions are covered, validated, and regression-protected through automated tests.

**🎯 Acceptance Criteria**

* Test suite implemented for all 9 FSM states:
  + INIT, DISCOVERY, LEARN, DECIDE, ENFORCE, OBSERVE, FEEDBACK, CORRECT, EXIT
* Valid state transitions are tested end-to-end:
  + e.g., INIT → DISCOVERY → LEARN → DECIDE → ENFORCE
* Invalid transitions (e.g., ENFORCE → INIT, EXIT → LEARN) are rejected and logged
* Stub FSM simulation engine allows triggering transitions via synthetic events
* Tests include:
  + Success paths
  + Blocked paths
  + Edge cases (rapid transition, duplicate events, missing context)
* All test logs use structured format via logger (T1-08)
* Results are integrated into CI (triggered by T6-09)
* Test plan and usage documented in test/fsm/README.md

**🧩 Notes**

* Forms the baseline for FSM coverage validation in every phase
* Required before simulating zone-level FSM behavior (T2-09) and AI-based transitions (T3+)
* Future versions will simulate distributed FSM instances across multiple ATZs

**🧾 User Story: T2-09 – Simulate ATZ Formation + Split**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-05 – Implement ATZ Zone Formation  
**Predecessor Status:** 🔵 **To Do**

**As a** Test/Simulation Engineer, **I want to** simulate dynamic ATZ behavior during formation and zone split. **So that** we can validate the ATROP FSM's responsiveness to topology changes and isolation events triggered by anomalies.

**🎯 Acceptance Criteria**

* A simulation scenario is created that includes:
  + A new node joining the network
  + Zone formation logic triggering assignment to a new or existing ATZ
  + Anomaly or failure causing a zone split (e.g., trust score drop or unreachable neighbor)
* FSM behavior is tracked during:
  + Join event (INIT → DISCOVERY → LEARN)
  + Zone assignment (DISCOVERY → LEARN)
  + Split/isolated fallback (ENFORCE → CORRECT → ISOLATE)
* Zone reformation is observed with updated zone\_id mapping
* Logs capture transitions, zone changes, and anomaly reaction
* Test results are deterministic and validated via assertions or CI reporting
* Scenario is documented in test/simulation/atz\_split\_test.py|cpp
* Use case added to docs/fsm/zones.md under "Simulated Zone Behavior"

**🧩 Notes**

* Completes the testing loop for ATZ zone logic started in T2-05
* Ties into FSM transition validation (T2-08) and feedback-triggered correction handling (T2-07)
* Early signal for distributed FSM behavior needed in Phase 7 (T7-06) and Phase 8 (T8-04)

**🧾 User Story: T2-10 – Document Protocol Flow**

**Story Points:** **2**  
**Status:** 🔵 **To Do**  
**Predecessor:** All Phase 2 Tasks  
**Predecessor Status:** In Progress

**As a** Documentation SME, **I want to** create visual and written documentation of the ATROP protocol flow. **So that** contributors, vendors, and reviewers can clearly understand how packets traverse the FSM and influence state transitions.

**🎯 Acceptance Criteria**

* At least one full-sequence diagram is created showing:
  + Incoming packet → Dispatcher → Parsed → FSM transition → State action → Outbound response
* Sequence diagrams cover different message types:
  + Discovery
  + Decision
  + Correction
* Transitions across FSM states are shown clearly (e.g., DISCOVERY → LEARN → DECIDE → ENFORCE)
* Events triggering transitions are labeled (e.g., policy match, anomaly detected, SLA breached)
* All diagrams are generated using PlantUML or an equivalent tool and saved to docs/diagrams/
* Textual explanation of each flow included in docs/protocol/protocol\_flow.md
* Visuals reviewed and approved by Protocol Architect for accuracy

**🧩 Notes**

* Completes the documentation layer for all state-driven behavior in Phase 2
* Required for internal knowledge sharing and external vendor onboarding
* Will be referenced by AI model inference flow (Phase 3) and integration test scenarios (Phase 7)

**🧩 Epic: ATROP Phase 3 – AI Model Engine, Intent Processing & Policy Decision**

**📘 Description**

This epic delivers the intelligence core of the ATROP control plane by integrating AI-based route inference, intent translation, and policy decision orchestration. It includes the construction of the AI model engine (GNN or RL), topology analytics, policy enforcement, and confidence fallback logic. Feedback loops from the data plane are also connected to drive adaptive routing based on real-time learning signals.

**🎯 Objectives**

* Implement the AI model engine to autonomously infer optimal routes
* Construct a live topology graph that supports zone-aware and metric-sensitive analytics
* Build the Intent Processing Unit (IPU) to translate service-level requests into actionable policy rules
* Integrate telemetry feedback from the ML data plane to improve model accuracy and reactivity
* Apply decision logic and trust scoring before injecting final route paths into the protocol
* Validate model output with policy awareness and rollback under low confidence scenarios
* Simulate federated learning behavior across zones to align with ATZ autonomy
* Document AI-driven routing lifecycle and inference path

**📦 Epic Deliverables**

* ✅ AI Model Engine (GNN or RL-based) capable of computing route decisions
* ✅ Topology Analytics Engine maintaining real-time graph of nodes, links, and metrics
* ✅ Intent Processing Unit (IPU) to parse and apply user intent into policy context
* ✅ Feedback Loop Receiver parsing telemetry (FIF, PIV) from the ML data plane
* ✅ Decision Policy Orchestrator applying trust scoring and injecting control-plane updates
* ✅ Confidence & fallback logic with safe-mode routing on model failure
* ✅ Federated training simulation across ATZs with gradient sync to mock coordinator
* ✅ Unit test suite for all AI modules (inference, mapping, feedback)
* ✅ Simulation of AI behavior under different policies (video, secure, bulk)
* ✅ Complete AI routing lifecycle diagram documented and committed

**🧾 User Story: T3-01 – Build AI Model Engine**

**Story Points:** **8**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T2-01 – Define Core ATROP States → ✅ Done
* T2-02 – Implement FSM Transitions → ✅ Done

**As an** AI Engineer, **I want to** implement the core AI-based model engine for the ATROP control plane. **So that** it can compute optimized routing decisions using graph topology and observed metrics through inference.

**🎯 Acceptance Criteria**

* A model engine is built using either:
  + Graph Neural Network (GNN) architecture
  + Reinforcement Learning (RL) with environment abstraction
* Input features include:
  + Node and link metrics from ATZ topology graph
  + Flow intents and constraints from IPU (T3-03)
  + Trust scores, queue depth, or latency where available
* Output is a route/path recommendation or zone-level forwarding decision
* Model operates in two modes:
  + Inference-only
  + Inference + self-learning (mock training environment)
* Inference latency and decision accuracy are logged and exportable
* Hooks provided for integration with:
  + Feedback signals (T3-04)
  + Policy filters (T3-05)
* Unit-tested with mock topology graphs and known route solutions
* Documented in models/ai\_engine/README.md with sample configs and usage

**🧩 Notes**

* Core component for all subsequent AI policy logic and route orchestration
* Will be extended in T3-06 for confidence scoring and in T3-08 for federated simulation
* Model structure should be lightweight and embeddable or remotely callable (for edge deployment paths)

**🧾 User Story: T3-02 – Topology Analytics Engine**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-05 – Implement ATZ Zone Formation  
**Predecessor Status:** 🔵 **To Do**

**As an** AI/Graph Developer, **I want to** build a module that maintains a real-time graph of ATZ topology state. **So that** the AI engine and control logic have accurate, queryable access to nodes, links, and metrics across the autonomous zone.

**🎯 Acceptance Criteria**

* A graph data structure is implemented using a suitable library or custom class:
  + Support for dynamic updates (node join/leave, link up/down)
  + Edge and node attributes (latency, utilization, zone, trust score)
* Graph APIs support:
  + Path lookup, cost calculation, zone boundary awareness
  + Node/edge tagging and metadata updates
  + Export to model engine input format
* Data sources may include:
  + FSM observations (T2-01)
  + Feedback packets (FIF/PIV from T3-04)
  + Static test input or mock telemetry
* All changes logged via the unified logger (T1-08)
* Unit tests cover:
  + Graph build/update logic
  + Edge cases (loop, partition, invalid update)
* Module documented in models/topology\_engine/README.md with schema diagram

**🧩 Notes**

* Required for inference input in T3-01 and federated learning simulation in T3-08
* Topology module must be modular and pluggable into AI or FSM workflows
* Future versions may integrate with live routing tables or SR/MPLS overlays in Phase 7+

**🧾 User Story: T3-03 – Intent Processing Unit (IPU)**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-06 – Embed Policy Awareness Hooks  
**Predecessor Status:** 🔵 **To Do**

**As an** Intent Architect. **I want to** implement an Intent Processing Unit (IPU), **So that** ATROP can translate high-level service or user intents into structured policy constraints for decision-making and enforcement.

**🎯 Acceptance Criteria**

* IPU module is created to accept and parse intent definitions:
  + SLA requirements (latency, jitter, bandwidth)
  + Security levels (trusted path only, encrypted transport)
  + Topological preferences (zone inclusion/exclusion, path diversity)
* Inputs may come from:
  + Static config files
  + CLI or API endpoint (future Phase 9 integration)
* Output is a structured policy object used by:
  + FSM transition gating (T2-06)
  + AI Model Engine (T3-01)
  + Decision Orchestrator (T3-05)
* Supports basic policy types:
  + Hard constraints (must match)
  + Soft preferences (weighted scoring)
* Logs policy parsing results and alerts for malformed or unsupported intents
* Includes validation rules and mock policy tests
* Usage documented in docs/policy/intent\_processing.md with example schemas

**🧩 Notes**

* Foundation for intent-based routing simulations (T3-09, T7-03)
* Will connect to external policy managers via API in Phase 9
* Enables dynamic service behavior beyond static routes or configs

**🧾 User Story: T3-04 – Feedback Loop Receiver**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T2-08 – Build FSM Unit Test Suite  
**Predecessor Status:** 🔵 **To Do**

**As a** Feedback Engineer, **I want to** implement a Feedback Loop Receiver. **So that** telemetry from the ML data plane (e.g., PIV and FIF packets) can be parsed and used to influence AI model updates or FSM corrective behavior.

**🎯 Acceptance Criteria**

* Receiver module created to parse the following telemetry formats:
  + **PIV** – Path Inference Vector
  + **FIF** – Feedback Injection Format
* Module supports:
  + Packet extraction from input buffer or IPC queue
  + Header and payload parsing (from T2-04 definitions)
  + Timestamping, flow ID correlation, anomaly tagging
* Parsed feedback is dispatched to:
  + AI model engine (T3-01) for learning updates
  + FSM correction handler (T2-07) for immediate fallback
* Receiver supports batching or streaming modes
* Structured logging of received feedback using T1-08 logger
* Unit tests include simulated FIF/PIV input and state validation
* Documented in docs/telemetry/feedback\_receiver.md with format examples

**🧩 Notes**

* Core component for ATROP’s closed-loop learning model
* Will be extended by T4-04 (FIF generation) and validated in T7-02 integration testing
* Should gracefully handle malformed or delayed packets to preserve model integrity

**🧾 User Story: T3-05 – Decision Policy Orchestrator**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T3-01 – Build AI Model Engine → 🔵 To Do
* T3-03 – Intent Processing Unit (IPU) → 🔵 To Do

**As a** Protocol/AI Lead, **I want to** implement a Decision Policy Orchestrator. **So that** the AI model’s routing decisions are validated against active policies and trust scores before being applied to the control plane.

**🎯 Acceptance Criteria**

* Module accepts candidate route decisions from the AI engine (T3-01)
* Applies policy filters derived from IPU (T3-03):
  + SLA, intent, zone preference, flow classification
* Applies dynamic trust scoring per link or path:
  + Inputs may include trust weights, security flags, anomaly counts
* Final decision is:
  + Accepted and injected into FSM/forwarding pipeline
  + Modified to meet policy thresholds
  + Rejected and triggers fallback or correction (hooks to T3-06, T2-07)
* All decisions logged with full traceability: source intent, AI rank, trust filter, final outcome
* Outputs injected into FSM DECIDE → ENFORCE transition scope
* Unit tests simulate conflicting or aligned inputs and assert correct decision behavior
* Documented in docs/ai/decision\_orchestrator.md with flow diagrams

**🧩 Notes**

* Enables full policy-aligned, AI-driven routing
* Required for simulations in T3-09 and diagramming in T3-10
* Serves as the policy-enforcing bridge between intelligent inference and protocol execution

**🧾 User Story: T3-06 – Model Confidence + Fallback**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T3-01 – Build AI Model Engine  
**Predecessor Status:** 🔵 To Do

**As an** AI Safety Engineer, **I want to** implement confidence scoring and fallback mechanisms for AI decisions. **So that** the system can detect low-confidence outputs and safely revert to baseline or policy-compliant behavior.

**🎯 Acceptance Criteria**

* AI model output includes a confidence\_score (e.g., 0.0–1.0) per decision
* Confidence threshold is configurable via central config (T1-07)
* If score falls below threshold:
  + Decision is either discarded, rerouted, or flagged
  + Fallback behavior is triggered:
    - Default/static route
    - Policy-prescribed safe mode (e.g., least-cost, trusted path)
    - FSM rollback to LEARN or transition to CORRECT
* Logging includes reason, score, and fallback path taken
* Safety behavior validated via:
  + Mock low-confidence inputs
  + Unit test assertion for fallback trigger
* Confidence logic integrated into T3-05 (Decision Orchestrator)
* Documented in docs/ai/safety\_fallback.md with scoring policy and override modes

**🧩 Notes**

* Critical for deployment in untrained or novel topologies
* Prepares model infrastructure for production-safe AI in later phases
* Integrates tightly with simulation and testing in T3-09 and fallback hooks in T4-06

**🧾 User Story: T3-07 – Unit Tests for AI Components**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T3-01 – Build AI Model Engine → 🔵 To Do
* T3-04 – Feedback Loop Receiver → 🔵 To Do

**As a** Test Engineer, **I want to** develop a set of unit tests for core AI components in ATROP. **So that** inference, intent mapping, and feedback processing behaviors can be validated independently and automatically.

**🎯 Acceptance Criteria**

* Unit tests are created for:
  + **AI Model Engine (T3-01)** – Inference logic, input/output validation, topology scenarios
  + **Intent Processing Unit (T3-03)** – Parsing, constraint enforcement, policy construction
  + **Feedback Receiver (T3-04)** – PIV/FIF parsing, anomaly injection, learning triggers
* Tests include:
  + Valid and invalid input cases
  + Model confidence thresholds and fallback paths
  + Policy match/mismatch behavior
* Tests are organized under test/ai/, executable via CI pipeline
* Assertions include:
  + Inference result integrity
  + Feedback propagation correctness
  + Policy compliance validation
* All test logs are integrated with logger (T1-08) and produce CI output
* Documented in test/ai/README.md with how-to-run and structure

**🧩 Notes**

* Enables regression coverage and AI model safety validation
* Required to unlock simulation and routing tests in T3-09 and T7-02
* Will later expand into integration test coverage in Phase 7+

**🧾 User Story: T3-08 – Federated Training Mock**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T3-01 – Build AI Model Engine  
**Predecessor Status:** 🔵 To Do

**As an** AI Researcher, **I want to** simulate a federated training environment across ATZs. **So that** each zone can train locally and share model gradients with a central (mock) controller, emulating decentralized learning behavior.

**🎯 Acceptance Criteria**

* Lightweight federated learning simulation is implemented:
  + Local training mock at each ATZ instance
  + Periodic sync of gradient deltas or model weights to a mock controller
  + Aggregation and redistribution of updated model parameters
* Supports configurable training intervals and convergence cycles
* All operations logged per zone (training start/end, sync event, update applied)
* Supports basic optimization (e.g., SGD, averaging) using dummy data
* Simulated zones identified by zone\_id from T2-05
* Unit tests simulate sync failure, drift, and recovery
* Documented in models/federated/README.md with example ATZ training map

**🧩 Notes**

* Builds foundation for real federated learning implementation in future phases
* Prepares ATROP for scalability in multi-domain and edge/metro deployments
* Links conceptually with AI fallback (T3-06) and telemetry-driven adaptation (T3-04)

**🧾 User Story: T3-09 – Policy-Driven Routing Simulation**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T3-03 – Intent Processing Unit (IPU) → 🔵 To Do
* T3-05 – Decision Policy Orchestrator → 🔵 To Do

**As an** AI/Test Engineer, **I want to** simulate AI-based routing decisions under various policy intents. **So that** we can validate how the model and policy layers respond to different traffic types and service requirements.

**🎯 Acceptance Criteria**

* Simulation scenarios created for key intent profiles:
  + Video (low-latency, high-bandwidth)
  + Bulk data (loss-tolerant, throughput-focused)
  + Secure (trusted links only, encrypted paths)
* Each scenario includes:
  + Mock flow injection
  + Intent-to-policy mapping via IPU
  + Model inference with or without fallback
  + Final decision trace via orchestrator (T3-05)
* Logs include:
  + Input intent
  + Candidate paths
  + Filtered decision
  + Policy acceptance/failure
* Assertions validate path compliance with original intent
* Results stored under test/simulation/policy\_routing/ with README.md summary

**🧩 Notes**

* Validates tight integration of model outputs, intent parsing, and policy logic
* Aligns directly with FSM DECIDE → ENFORCE phase and later integration tests in T7-03
* Demonstrates AI explainability for ATROP control decisions under diverse SLAs

**🧾 User Story: T3-10 – Document AI Routing Flow**

**Story Points:** **2**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T3-01 – Build AI Model Engine → 🔵 To Do
* T3-05 – Decision Policy Orchestrator → 🔵 To Do

**As a** Documentation SME, **I want to** create a visual and written overview of the AI-driven routing lifecycle. **So that** developers and reviewers can clearly understand how topology changes and feedback influence final route decisions in ATROP.

**🎯 Acceptance Criteria**

* A full-sequence diagram or flowchart is created showing:
  + Topology event (e.g., link down, zone change)
  + Telemetry/feedback received (FIF, PIV)
  + AI model inference and policy filtering
  + Final decision injection into FSM/forwarding layer
* Covers interactions between:
  + Topology Analytics Engine (T3-02)
  + Feedback Loop Receiver (T3-04)
  + Intent Processing Unit (T3-03)
  + AI Model (T3-01)
  + Decision Orchestrator (T3-05)
* Diagram created in PlantUML or similar and saved under docs/diagrams/ai/
* Textual explanation included in docs/ai/routing\_flow.md
* Reviewed and approved by AI/Protocol leads for accuracy and completeness

**🧩 Notes**

* Required for traceability, vendor onboarding, and simulation verification
* Will be reused in Phase 7+ integration scenarios and external documentation sets
* Helps bridge FSM, AI, and policy narratives across the ATROP architecture

**🧩 Epic: ATROP Phase 4 – ML Inference Agent and Telemetry Path (Data Plane Intelligence)**

**📘 Description**

This epic focuses on building the real-time ML inference engine that operates in the ATROP data plane. It includes the design of a lightweight classification agent, per-flow trust scoring, telemetry emission via FIF packets, and runtime model handling. The outcome is a self-observing data plane capable of influencing the control plane through learned behavior and feedback, enabling adaptive routing, safety enforcement, and brownfield deployment readiness.

**🎯 Objectives**

* Build a daemon that performs lightweight ML inference on live flow traffic
* Classify traffic patterns to support routing decisions (e.g., bursty vs. real-time flows)
* Emit structured telemetry using the FIF protocol for closed-loop feedback
* Assign dynamic trust scores to links and flows for security-aware forwarding
* Allow model hot-swapping and fallback for constrained environments
* Connect inference engine to packet visibility hooks (Netfilter, eBPF, IPTables)
* Validate interaction between data plane and control plane AI via telemetry
* Support brownfield testing using passive (non-enforcing) inference mode
* Document the entire ML feedback and inference lifecycle

**📦 Epic Deliverables**

* ✅ ML Inference Agent daemon implemented with runtime classification logic
* ✅ Flow Classification Model (DT/CNN) trained or fine-tuned on simulated traffic patterns
* ✅ Trust Score Engine for per-link/flow scoring based on dynamic conditions
* ✅ FIF telemetry generator that emits structured feedback from flow-level insight
* ✅ Runtime Model Loader API for swapping or reloading model versions on the fly
* ✅ Safety hooks that detect overload or low-confidence conditions and trigger bypass
* ✅ Packet visibility tap via Linux Netfilter/IPTables or eBPF integration
* ✅ Simulations covering video, bulk, bursty traffic types and corresponding ML reactions
* ✅ Unit and integration test coverage for inference, feedback, and response handling
* ✅ Brownfield “shadow mode” for passive inference in legacy deployments
* ✅ Documentation of end-to-end ML feedback loop with diagrams and examples

**🧾 User Story: T4-01 – ML Inference Agent**

**Story Points:** **8**  
**Status:** 🔵 **To Do**  
**Predecessor:** T3-01 – Build AI Model Engine  
**Predecessor Status:** 🔵 To Do

**As an** ML Engineer, **I want to** develop a real-time inference daemon for the data plane. **So that** it can classify live network flows using a lightweight ML model and support adaptive, intelligent forwarding decisions in ATROP.

**🎯 Acceptance Criteria**

* Daemon is implemented (e.g., inference\_agent.cpp / inference\_agent.py)
* Uses a lightweight ML model such as:
  + Decision Tree (DT)
  + Convolutional Neural Network (CNN)
* Capable of:
  + Processing real-time flow metadata (packet headers, timing, rate)
  + Outputting classification labels (e.g., video, bulk, bursty, anomaly)
  + Assigning local decisions or triggering FIF packet generation (T4-04)
* Modular design supporting:
  + Model loading interface (T4-05)
  + Trust score integration (T4-03)
  + Safety fallback logic (T4-06)
* Can be run on edge devices with CPU/memory constraints
* Logging is integrated using ATROP’s structured logger (T1-08)
* Includes test hooks for offline and live data classification
* Documented in docs/dataplane/inference\_agent.md with flow chart and usage

**🧩 Notes**

* Foundation for telemetry feedback, trust scoring, and control-plane interaction
* Must support later simulation and shadow testing scenarios (T4-08, T4-11)
* Expected to operate continuously on each node participating in ATROP's data plane

**🧾 User Story: T4-02 – Flow Classification Model**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Data Scientist, **I want to** implement or fine-tune a flow classification model. **So that** the ML inference agent can accurately label traffic by type and behavioral pattern (e.g., congestion, latency, burstiness) in real time.

**🎯 Acceptance Criteria**

* A supervised ML model is selected and trained using mock or real flow datasets:
  + Decision Tree (DT), CNN, or other lightweight model
* Input features may include:
  + Packet arrival time deltas
  + Flow duration and byte rate
  + TCP flags, packet size variance
* Output labels must include:
  + Application type (e.g., video, bulk, VoIP, malicious)
  + Congestion pattern (steady, bursty)
  + Latency sensitivity level (high, low)
* Model is serialized for runtime inference (e.g., joblib, ONNX, TorchScript)
* Evaluation metrics (accuracy, precision, recall) logged and documented
* Inference benchmarked under constrained CPU/memory conditions
* Integrated into the ML Inference Agent (T4-01) with test pass
* Documented in models/classification/README.md with data schema and performance

**🧩 Notes**

* Required to power real-time data plane telemetry (T4-04) and trust scoring (T4-03)
* Model must remain efficient enough for deployment in brownfield or edge setups
* Enables simulation replay (T4-08) and full feedback loop coverage (T4-10)

**🧾 User Story: T4-03 – Flow Trust Score Engine**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-02 – Flow Classification Model  
**Predecessor Status:** 🔵 To Do

**As an** ML Engineer, **I want to** implement a trust scoring engine that evaluates flows and links. **So that** ATROP can make forwarding decisions that account for security posture, behavioral anomalies, and path reliability in real time.

**🎯 Acceptance Criteria**

* Trust score module implemented as a component within the ML inference agent (T4-01)
* Trust scores calculated based on:
  + Flow classification result (from T4-02)
  + Historical anomaly rate, packet drop rate, latency/jitter
  + Security triggers (e.g., known bad actor, zone policy)
* Scores are:
  + Normalized (e.g., 0.0–1.0 or 0–100)
  + Time-sensitive (decay logic or exponential moving average)
  + Zone-aware (scores may differ per ATZ scope)
* Output includes:
  + Trust score per flow
  + Trust weight per link/interface
* Logging integrated with structured telemetry format
* Outputs optionally feed into:
  + FIF packet generator (T4-04)
  + Control plane policy orchestrator (T3-05)
* Unit tests simulate changes in behavior and verify trust score trends
* Documented in docs/trust/trust\_engine.md with examples and scoring logic

**🧩 Notes**

* Enables security-aware routing in both control and data planes
* Trust score can become a core metric in feedback, policy enforcement, and fallback routing
* Future extensions may include zone-wide trust aggregation or cross-domain threat signaling

**🧾 User Story: T4-04 – Feedback Injection Generator**

**Story Points:** **4**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Telemetry Developer, **I want to** implement a module that generates Feedback Injection Format (FIF) packets. **So that** observations from the ML inference agent can be relayed to the control plane for adaptive decision-making and model correction.

**🎯 Acceptance Criteria**

* FIF packet format implemented according to ATROP spec:
  + Includes fields such as: flow ID, label, trust score, observed anomaly, zone ID, timestamp
* Generator module:
  + Collects classified flow output from ML inference agent (T4-01)
  + Pulls trust score metadata (from T4-03)
  + Serializes and emits telemetry packets at fixed intervals or on significant event
* Packets are:
  + Routed via IPC or direct socket to control plane receiver (T3-04)
  + Logged with debug and trace level visibility
* FIF emission supports:
  + Configurable thresholds (anomaly only, top-N flows, periodic mode)
  + Graceful degradation when queue/backpressure detected
* Includes test scenarios for well-formed and malformed FIF payloads
* Sample output documented in docs/telemetry/fif\_format.md with annotated example

**🧩 Notes**

* Enables feedback loop from data plane to control logic for reinforcement learning or FSM fallback
* Must work efficiently under real-time load and constrained environments
* Required for full integration tests in T4-10 and ML-to-AI sync validation in Phase 7

**🧾 User Story: T4-05 – Model Loader API**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As an** Infrastructure Developer, **I want to** implement a Model Loader API. **So that** the ML inference agent can load or hot-swap new model versions at runtime without restarting or disrupting flow classification.

**🎯 Acceptance Criteria**

* Loader interface added to the ML Inference Agent (T4-01):
  + Supports load\_model(path) or reload\_model(version) operations
  + Validates model format, version, and schema before loading
* Compatible with common serialization types:
  + joblib, ONNX, TorchScript, etc.
* Loader logs:
  + Load success/failure
  + Versioning info and model hash
  + Timestamp of operation
* Optional config flags:
  + Auto-reload on config/model change
  + Manual reload via CLI/API call
* Includes fallback to previous model on load failure
* Unit tests for:
  + Valid model swap
  + Version mismatch
  + Corrupted model handling
* Usage documented in docs/dataplane/model\_loader.md with CLI/API examples

**🧩 Notes**

* Enables continuous model refinement and federated sync (T3-08)
* Required for runtime resilience and integration with brownfield deployments (T4-11)
* Supports edge-node model optimization and gradual rollout strategies

**🧾 User Story: T4-06 – Safety Fallback Hooks**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Safety Engineer, **I want to** implement safety fallback hooks in the ML inference agent. **So that** the system can bypass or disable inference when resource limits are exceeded or confidence levels are too low, ensuring safe operation under load.

**🎯 Acceptance Criteria**

* Runtime monitors implemented for:
  + CPU usage
  + Memory usage
  + Inference confidence (from T3-06 structure)
* If thresholds are breached:
  + Inference step is skipped or bypassed
  + Flow is tagged with a default label (e.g., “unclassified”)
  + Optional logging and alert is triggered
* Supports fallback actions:
  + Pass-through forwarding
  + Static policy route
  + Deferred FIF generation or safe flagging
* Configurable thresholds via T1-07 central config loader
* Unit tests simulate overload and confirm bypass behavior
* Documented in docs/dataplane/safety\_fallback.md with flowchart and policy examples

**🧩 Notes**

* Ensures ML agent remains stable in edge or degraded conditions
* Required for resilience in brownfield environments and constrained hardware
* Complements T3-06 control-plane fallback with local agent self-protection

**🧾 User Story: T4-07 – Inline Flow Tap Hook**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Platform Developer  
**I want to** implement a low-level packet tap hook using Netfilter/IPTables or eBPF  
**So that** the ATROP ML inference agent can observe live flow characteristics at the packet level in real time without disrupting forwarding.

**🎯 Acceptance Criteria**

* A packet capture/tap mechanism is deployed using one of:
  + **Netfilter Queue (NFQUEUE)**
  + **eBPF/XDP** program
  + **IPTables raw/mangle hook**
* Captured packets (or metadata) are forwarded to the inference agent:
  + Flow-level summarization (5-tuple, byte count, packet timing)
  + Sampling or full-stream mode selectable via config
* Zero-copy and minimal overhead processing validated
* Tap supports filtering by:
  + Interface, zone, or protocol
  + Flow type (TCP/UDP), port, DSCP
* Captured flow context is forwarded to:
  + T4-02 Flow Classifier
  + T4-03 Trust Engine
* Includes stress/load test under traffic simulation
* Hook integration documented in docs/dataplane/flow\_tap.md with setup instructions and diagrams

**🧩 Notes**

* Required for real-time flow inference in production or lab simulation
* Must support both inline and passive (T4-11) deployment options
* Enables testing and replay scenarios in T4-08 and visibility for T7-08

**🧾 User Story: T4-08 – Flow Behavior Simulation**

**Story Points:** **4**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T4-01 – ML Inference Agent → 🔵 To Do
* T4-02 – Flow Classification Model → 🔵 To Do

**As a** Test Engineer, **I want to** simulate different types of network flows. **So that** I can validate the ML inference agent’s classification accuracy and ensure correct FIF telemetry is generated and sent upstream.

**🎯 Acceptance Criteria**

* Simulated flow types include:
  + **Video** (high throughput, consistent rate)
  + **Bulk data** (bursty, high-volume)
  + **VoIP/real-time** (low latency, low jitter)
  + **Malicious or anomaly patterns** (optional)
* Simulation tools or scripts created using:
  + tc, iperf, hping3, or custom Python generator
* Each flow triggers:
  + Classification by the ML model (T4-02)
  + FIF emission by the feedback injector (T4-04)
* FIF output inspected to confirm:
  + Correct label
  + Trust score (if T4-03 is implemented)
  + Timely telemetry generation
* Logs trace classification and telemetry lifecycle
* Simulation scenarios documented in test/dataplane/simulations/README.md
* Scripts support automated execution with deterministic outputs

**🧩 Notes**

* Validates entire data-plane ML feedback loop from packet → inference → feedback
* Required for testing integration with T4-10 and later routing reactions in Phase 7
* Helps tune thresholds for T4-06 (Safety Fallback) and model improvement in Phase 3

**🧾 User Story: T4-09 – Unit Tests (Inference Path)**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Test Engineer, **I want to** write unit tests that validate the end-to-end inference path. **So that** we can ensure flows are correctly vectorized, classified by the model, and lead to proper FIF telemetry generation.

**🎯 Acceptance Criteria**

* Unit tests are implemented for:
  + Flow vectorization logic (conversion to model-ready format)
  + Model inference call using a stub or real model (T4-02)
  + FIF telemetry construction logic (T4-04)
* Tests include:
  + Valid flow input → expected classification → telemetry payload
  + Malformed flow input → graceful fallback or safe labeling
  + Model fallback trigger (if low confidence or exception raised)
* Mocks or fixtures simulate real packet/flow metadata
* Output FIF packets validated against expected schema
* Tests integrated with CI (see T6-09), and logs verified
* Documented under test/dataplane/unit/inference/README.md with structure and execution instructions

**🧩 Notes**

* Ensures inference logic works independently before full system integration
* Complements behavioral simulation tests (T4-08) and helps detect regressions
* Also prepares foundation for integration testing with control plane (T4-10)

**🧾 User Story: T4-10 – Integration Test with Control Plane**

**Story Points:** **5**  
**Status:** 🔵 **To Do**  
**Predecessors:**

* T4-04 – Feedback Injection Generator → 🔵 To Do
* T3-04 – Feedback Loop Receiver → 🔵 To Do

**As an** Integration Lead, **I want to** validate the end-to-end feedback interaction between the ML data plane and AI control plane. **So that** we can ensure the control plane responds appropriately to FIF telemetry and adjusts routing decisions based on real-time flow behavior.

**🎯 Acceptance Criteria**

* Integration test environment set up with:
  + Active ML Inference Agent (T4-01) emitting FIF packets
  + Control plane feedback receiver (T3-04) consuming and processing FIF
  + AI model reacting to changes (T3-01)
* Test cases include:
  + Flow classified as bursty/congested → feedback sent → route change or FSM correction
  + Trust score degraded → alternate path or fallback triggered
  + No reaction on malformed or low-confidence FIF
* Logs confirm:
  + FIF received and parsed correctly
  + AI model adjusted or policy applied
  + State machine or route decision changed
* FIF telemetry integrity checked across IPC or socket layer
* Documented in test/integration/dataplane\_to\_control/README.md with diagrams, commands, and expected results

**🧩 Notes**

* Completes the ML→AI feedback loop, a core innovation of ATROP
* Builds on top of unit/behavioral tests (T4-09, T3-07) and prepares for full stack emulation in Phase 8
* Demonstrates real-time control reactivity and closes the autonomous decision-making loop

**🧾 User Story: T4-11 – Brownfield Shadow Mode**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As a** Compatibility Developer, **I want to** implement a passive shadow mode for the ML inference agent. **So that** it can observe and classify flows without enforcing decisions or emitting control feedback, enabling safe deployment in legacy (brownfield) environments.

**🎯 Acceptance Criteria**

* Shadow mode implemented as a configurable flag or runtime argument
* In this mode:
  + Inference is performed normally
  + No routing decisions, trust score injection, or FSM transitions are triggered
  + FIF telemetry packets are optionally logged but not transmitted
* Results logged locally or to a separate monitoring channel for validation
* Must support toggling between shadow mode and active mode without restart
* Integration validated in an emulated legacy deployment scenario (e.g., FRR/OSPF co-existence)
* Documented in docs/compatibility/shadow\_mode.md with use cases, limitations, and activation guide

**🧩 Notes**

* Enables risk-free deployment of ATROP ML agents in existing infrastructure
* Required for vendor acceptance, brownfield adoption, and gradual rollout strategies
* Forms foundation for deployment planning and test scenarios in Phase 8 (T8-11, T8-09)

**🧾 User Story: T4-12 – Document ML Path & FIF**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T4-01 – ML Inference Agent → 🔵 To Do
* T4-04 – Feedback Injection Generator → 🔵 To Do

**As a** Documentation SME, **I want to** document the full ML data path and FIF telemetry flow. **So that** developers, integrators, and vendors can understand how packet observations drive feedback into the control plane for adaptive routing.

**🎯 Acceptance Criteria**

* One or more sequence diagrams created showing:
  + Flow enters data plane
  + Model classifies the flow
  + Trust score assigned
  + FIF packet generated
  + Feedback reaches AI control plane (T3-04)
* Each step clearly labeled with source module, output, and interface (IPC, socket, etc.)
* PlantUML or equivalent diagrams saved under docs/diagrams/dataplane/ml\_fif\_flow.puml
* Explanatory documentation added in docs/dataplane/ml\_path.md including:
  + Flow description
  + Data format summary
  + Configuration impact (e.g., threshold, bypass)
* Reviewed by ML and Telemetry leads for accuracy

**🧩 Notes**

* Completes visibility into one of the core closed-loop components of ATROP
* Required for onboarding, vendor review, and integration test planning in Phases 7–9
* Complements control-plane routing documentation from T3-10

**🧩 Epic: ATROP Phase 5 – Inter-Process Communication (IPC) Core Protocol & Messaging**

**📘 Description**

This epic defines and implements ATROP’s internal IPC communication fabric across the control plane, data plane, AI/ML engines, and auxiliary agents. It covers the full lifecycle from protocol specification and server/client architecture to message queuing, dynamic configuration, and security enforcement. The goal is to enable fast, reliable, secure, and extensible communication between autonomous ATROP modules using well-defined message formats.

**🎯 Objectives**

* Define the IPC messaging specification (e.g., protobuf, JSON schema) with strict field types and extensibility
* Implement the IPC server (daemon hub) for routing messages between agents
* Create client libraries for control plane, data plane, model inference agents, and monitoring tools
* Support asynchronous message delivery via shared memory or queue systems
* Enforce agent identity, lifecycle registration, and permission controls
* Enable dynamic runtime reconfiguration via IPC messages
* Establish robust unit/integration test coverage and performance baseline
* Provide complete documentation for message types, flow, and integration guidance

**📦 Epic Deliverables**

* ✅ IPC Protocol Spec with headers, message types, and transport formats (Protobuf/JSON)
* ✅ IPC Daemon Hub for message routing via UNIX sockets or gRPC
* ✅ Client libraries for sending/receiving protocol messages across ATROP agents
* ✅ Queue-based architecture for async FIF, policy, and feedback messaging
* ✅ Agent lifecycle protocol supporting registration, heartbeats, and version sync
* ✅ Hot-reload of models/configurations/policies via defined IPC message triggers
* ✅ IPC Security Layer with access control, authentication, and socket permissions
* ✅ Unit tests for protocol messages and edge cases (invalid headers, timeouts)
* ✅ End-to-end integration testbed validating IPC across functional modules
* ✅ Uptime and identity tracker to maintain process health metadata
* ✅ Documentation of the IPC interface, message catalog, examples, and sequence flows

**🧾 User Story: T5-01 – Define IPC Protocol Spec**

**Story Points:** **3**  
**Status:** 🔵 **To Do**  
**Predecessor:** Phase 0 Initialization  
**Predecessor Status:** ✅ Done

**As a** Protocol Engineer, **I want to** define the IPC protocol specification used by ATROP internal components. **So that** control, data, AI/ML, and telemetry modules can communicate reliably using a structured and extensible message format.

**🎯 Acceptance Criteria**

* IPC protocol specification includes:
  + Standard header fields (e.g., msg\_type, src, dst, timestamp, zone\_id)
  + Message body schemas for:
    - FIF telemetry
    - Policy injection
    - Model updates
    - Config reload
    - FSM control events
* Message formats designed in:
  + Protobuf (.proto files) or
  + JSON Schema (.schema.json)
* Support for:
  + Versioning
  + Optional/required fields
  + Extensibility for future message types
* Includes specification of transport constraints (e.g., UNIX socket framing, gRPC, TLS wrapping)
* Sample messages included for each type with field-by-field explanation
* Validated against basic parser implementation or schema compiler
* Documented in docs/ipc/ipc\_protocol\_spec.md and ipc/proto/ or ipc/schema/ directory

**🧩 Notes**

* Required to enable implementation of the IPC Server (T5-02), Clients (T5-03), and Testbed (T5-09)
* Must anticipate cross-platform use and embedded agent integration in later phases
* Forms foundation for configuration reload (T5-06), security (T5-07), and lifecycle tracking (T5-05, T5-10)

**🧾 User Story: T5-02 – Build IPC Server (Daemon Hub)**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T5-01 – Define IPC Protocol Spec  
**Predecessor Status:** 🔵 To Do

**As an** Infrastructure Developer, **I want to** build a central IPC server (daemon hub). **So that** all ATROP agents can exchange structured messages securely and efficiently through a unified communication fabric.

**🎯 Acceptance Criteria**

* IPC server implemented as a standalone daemon process (e.g., ipc\_hub)
* Supports:
  + **UNIX domain sockets** (default)
  + **gRPC or named pipe fallback** (optional for container/cross-host setups)
* Handles:
  + Agent registration and session tracking
  + Message routing between control plane, data plane, AI agents, etc.
  + Framing and parsing based on protocol spec (T5-01)
* Implements event-driven or async message loop (e.g., select(), epoll, asyncio)
* Can multiplex:
  + FIF telemetry
  + Model updates
  + Config reload triggers
  + FSM control packets
* Logs all transactions using structured logging with trace IDs
* Includes test stubs or mock clients for connectivity validation
* Codebase located under ipc/server/ with startup script and README

**🧩 Notes**

* Forms the communication backbone of ATROP across all autonomous modules
* Required for client development (T5-03), shared memory integration (T5-04), and config reload (T5-06)
* Later extended for heartbeat tracking (T5-10) and secure messaging (T5-07)

**🧾 User Story: T5-03 – Create IPC Clients**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T5-02 – Build IPC Server (Daemon Hub)  
**Predecessor Status:** 🔵 To Do

**As a** Core Developer, **I want to** implement IPC client libraries for key ATROP components. **So that** the control plane, data plane, and ML agents can send and receive protocol messages via the central IPC daemon.

**🎯 Acceptance Criteria**

* Client libraries created in:
  + **C++** for control plane and daemon agents
  + **Python** for AI/ML modules and simulation tools
* Clients support:
  + Socket connection and reconnection logic
  + Message framing and parsing based on T5-01 spec
  + Send/receive APIs with internal callback dispatching
  + Optional message queue buffering (send/retry logic)
* Client roles include:
  + **Control plane** → receives FIF, sends policies
  + **ML agent** → sends FIF, receives config updates
  + **Data plane** → receives model state, responds to FSM commands
* Includes mock mode for offline test injection
* Logging, metrics, and diagnostics integrated per client
* Unit tests verify:
  + Message send/receive roundtrip
  + Connection error handling
  + Spec compliance
* Client code placed under ipc/client/ with API usage examples and README

**🧩 Notes**

* Required for protocol flow completion between agents (T4-10, T3-04, T2-07)
* Forms the basis for all runtime interactions within ATROP
* Enables integration testing, heartbeat (T5-10), config reload (T5-06), and security scope (T5-07)

**🧾 User Story: T5-04 – Shared Memory or Queue System**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T5-01 – Define IPC Protocol Spec  
**Predecessor Status:** 🔵 To Do

**As a** Systems Engineer, **I want to** implement a shared memory or queue-based messaging system. **So that** ATROP agents can exchange FIF and decision messages asynchronously with low latency and minimal CPU overhead.

**🎯 Acceptance Criteria**

* A message queuing mechanism is implemented using one of:
  + multiprocessing.Queue or asyncio.Queue (Python)
  + Lockless ring buffer or shared memory (C++)
* Queues must support:
  + **FIF telemetry injection**
  + **AI/ML decision delivery**
  + **Policy/config updates**
* Features include:
  + Message serialization/deserialization (T5-01 format)
  + Bounded buffer size with overflow logging
  + Optional in-memory FIFO persistence
* Works in coordination with IPC Server (T5-02) for fan-out routing
* Queue modules abstracted and reusable across agents
* Stress-tested under high-throughput flow classification simulation (e.g., T4-08)
* Unit tests validate enqueue, dequeue, timeout, and overflow
* Implementation documented in docs/ipc/shared\_queue.md with diagrams and usage guide

**🧩 Notes**

* Enables decoupling of fast data-plane observations from control-plane processing
* Required for async, non-blocking feedback routing in real-time ATROP environments
* Complements heartbeat, reload triggers, and integration testbed (T5-09)

**🧾 User Story: T5-05 – Define Agent Lifecycle Protocol**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T5-02 – Build IPC Server (Daemon Hub)  
**Predecessor Status:** 🔵 To Do

**As an** Infra/Platform Lead, **I want to** define and implement the agent lifecycle protocol. **So that** all ATROP agents can register, advertise their capabilities, and negotiate version compatibility with the central IPC server.

**🎯 Acceptance Criteria**

* Protocol messages defined for:
  + agent\_register
  + agent\_heartbeat
  + agent\_capabilities
  + agent\_disconnect
* On connection, each agent must:
  + Send its ID, role (e.g., ml\_agent, fsm\_node, monitor), supported protocol version, and capabilities
  + Receive acknowledgment and assigned session token
* Version mismatch and unsupported roles logged and rejected with error code
* Heartbeat protocol supports:
  + Periodic liveness check
  + Last seen timestamp
  + Restart reason if applicable
* Capabilities include:
  + Supported message types
  + Optional features (e.g., config reload, hot-swap support)
* Stored in server-side registry for real-time lookup
* Documented in docs/ipc/agent\_lifecycle.md with sample message flow and schema

**🧩 Notes**

* Enables robust coordination and observability of distributed agents
* Required for heartbeat tracking (T5-10), secure session management (T5-07), and dynamic config push (T5-06)
* Essential for ATROP's multi-agent orchestration and future scalability

**🧾 User Story: T5-06 – Reloadable Config Handling**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T5-02 – Build IPC Server (Daemon Hub)  
**Predecessor Status:** 🔵 To Do

**As an** Infrastructure Developer, **I want to** implement IPC-triggered configuration reload handling. **So that** ATROP agents can update models, policies, and runtime parameters without restarts, improving flexibility and reducing downtime.

**🎯 Acceptance Criteria**

* Define IPC message type config\_reload with parameters:
  + target\_agent, config\_type, config\_path, timestamp, reason
* Agents listening via IPC client (T5-03) can receive and process reload requests
* Supported reload targets include:
  + AI/ML model file (T4-05)
  + Policy or intent rules
  + Logging level or threshold parameters
* Reloads are:
  + Logged with success/failure and reload version/hash
  + Gracefully handled (no crashes or in-flight message loss)
* Reload behavior is testable using mock messages or CLI triggers
* Documented in docs/ipc/config\_reload.md with use cases and sample schema

**🧩 Notes**

* Critical for supporting long-lived agents and dynamic updates in field deployments
* Used by orchestration or simulation tools in Phase 7/8 and tied to T5-03 client library logic
* Complements model loader (T4-05), lifecycle management (T5-05), and IPC security (T5-07)

**🧾 User Story: T5-07 – IPC Security Layer**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T5-01 – Define IPC Protocol Spec  
**Predecessor Status:** 🔵 To Do

**As a** Security Engineer, **I want to** implement a security layer for ATROP’s IPC system. **So that** all agents are authenticated, sockets are access-controlled, and identity is enforced to prevent spoofing or unauthorized message injection.

**🎯 Acceptance Criteria**

* Authentication mechanisms implemented for agent connection:
  + Agent identity token (generated at registration)
  + Role verification (ml\_agent, fsm\_node, control, etc.)
  + Optional TLS certificate validation (for gRPC mode)
* UNIX domain socket access controlled via:
  + File permissions
  + Group ownership (e.g., atrop-agents)
  + Runtime checks on UID/GID at connection time
* IPC message header includes signed agent ID and optional nonce
* Tampering or impersonation attempts logged with reason and source
* Fallback mode supports:
  + Local development bypass with warning
  + Read-only test clients
* Security policy file defines:
  + Trusted agents
  + Permitted message types per role
* Tests validate:
  + Unauthorized access is denied
  + Impersonation is detected
  + Secure and insecure modes function independently
* Documented in docs/ipc/security.md with examples and threat model

**🧩 Notes**

* Required for secure deployment in production and vendor-integrated environments
* Supports all downstream components (T5-02 server, T5-03 clients, T5-09 testbed)
* Forms baseline for secure telemetry exchange, model injection, and intent enforcement

**🧾 User Story: T5-08 – Unit Tests for IPC Protocol**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T5-01 – Define IPC Protocol Spec → 🔵 To Do
* T5-03 – Create IPC Clients → 🔵 To Do

**As a** Test Engineer, **I want to** write unit tests for the IPC protocol implementation. **So that** all defined message types and client interactions are validated for correctness, resilience, and spec compliance.

**🎯 Acceptance Criteria**

* Unit tests implemented for each IPC message type:
  + FIF telemetry
  + Model update
  + Config reload
  + Agent registration / heartbeat
  + Policy injection
* Test coverage includes:
  + Valid message round-trip (send → receive → parse)
  + Malformed input detection and rejection
  + Header validation (type, version, required fields)
  + Simulated reconnect scenarios and timeout handling
* IPC client and stub server used to emulate behavior
* All messages tested against T5-01 spec (JSON or Protobuf)
* Tests run in CI (see T6-09), and log assertions validated
* Test outputs documented under test/ipc/unit/ with README.md and sample cases

**🧩 Notes**

* Ensures IPC messaging stability across all agents before integration (T5-09)
* Detects regressions early and enforces schema compatibility
* Required foundation for production readiness and protocol auditability

**🧾 User Story: T5-09 – IPC Integration Testbed**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T5-04 – Shared Memory or Queue System → 🔵 To Do
* T5-06 – Reloadable Config Handling → 🔵 To Do

**As an** Integration Engineer, **I want to** build a complete IPC integration testbed. **So that** live ATROP agents can exchange messages like FIF telemetry, model updates, and config reloads in a fully coordinated runtime environment.

**🎯 Acceptance Criteria**

* Testbed includes at minimum:
  + ML Inference Agent (T4-01)
  + AI Engine or FSM receiver (T3-04, T2-01)
  + IPC Server (T5-02)
  + Client libraries (T5-03) connected to queue system (T5-04)
* End-to-end message types validated:
  + FIF from ML → control plane
  + Model updates from controller → ML agent
  + Config reloads between agents
* Transport options tested:
  + UNIX domain sockets
  + Optional gRPC fallback
* Queue handling verified for:
  + Message delay
  + Backpressure scenarios
  + Failover / restart continuity
* Observability hooks validate message path, latency, and errors
* Results and logs documented in test/ipc/integration/README.md with test flow diagrams

**🧩 Notes**

* Essential for validating cross-agent IPC reliability and coordination before rollout
* Enables lifecycle and safety scenarios (T5-05, T5-06, T5-10) to be tested in context
* Foundation for Phase 7 integration and Phase 8 staging simulations

**🧾 User Story: T5-10 – Daemon Identity and Uptime Tracker**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T5-05 – Define Agent Lifecycle Protocol  
**Predecessor Status:** 🔵 To Do

**As a** Core Engineer, **I want to** implement identity tracking and uptime monitoring for each ATROP daemon. **So that** the system can record agent status, version, and availability for observability, debugging, and resilience.

**🎯 Acceptance Criteria**

* On startup, each agent must:
  + Register with IPC using a unique agent\_id
  + Send software version, build\_hash, and start\_time
* Uptime tracker includes:
  + Continuous heartbeat timestamping
  + Duration since start
  + Last restart reason (if available)
* Agent status stored or reported via:
  + IPC Server registry (T5-02)
  + Optional /status CLI/API endpoint or debug log output
* Status includes:
  + Current role and zone
  + Active features (e.g., ML active, feedback enabled)
  + Last message sent/received timestamp
* Tracker logs warnings for missing heartbeats or rapid restarts
* Output documented in docs/ipc/daemon\_status.md with field schema and examples

**🧩 Notes**

* Enables visibility for operations, monitoring, and fault recovery (e.g., T10-05)
* Forms part of agent coordination lifecycle and readiness tracking
* Required for pre-deployment audits and staging simulations (Phase 8)

**🧾 User Story: T5-11 – Document IPC Interface & Messages**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T5-01 – Define IPC Protocol Spec
* T5-02 – Build IPC Server
* T5-03 – Create IPC Clients  
  **Predecessor Status:** 🔵 To Do

**As a** Documentation SME, **I want to** provide comprehensive documentation of the IPC interface and message structure. **So that** developers and integrators can implement, debug, and extend inter-agent messaging in ATROP with clarity and confidence.

**🎯 Acceptance Criteria**

* Message catalog created listing:
  + Each supported msg\_type
  + Required and optional fields
  + Valid value ranges and data types
* Sequence diagrams included showing:
  + Agent registration and handshake
  + FIF transmission from ML → Control
  + Config reload from Control → ML
  + Policy injection flow
* Security section outlines:
  + Authentication method
  + Socket/file permission rules
  + Role-based message validation
* File structure:
  + docs/ipc/protocol.md – message types and format
  + docs/ipc/sequence\_flows.md – interaction examples
  + docs/ipc/security\_model.md – trust, auth, and isolation
* Includes diagrams (PlantUML or equivalent) under docs/diagrams/ipc/

**🧩 Notes**

* Required for onboarding, vendor adoption, and cross-team development
* Complements all work in T5-01 through T5-10 and supports later Phase 6–8 test automation
* Ensures message format and behavior are consistent across teams and implementations

**🧩 Epic: ATROP Phase 6 – Unit Testing Infrastructure & Strategy**

**📘 Description**

This epic establishes the full-stack unit testing framework for the ATROP architecture. It includes language-specific infrastructure (Python/pytest and C++/gtest), module-aligned layout, protocol-specific test suites, CI automation, and documentation. The objective is to enable continuous validation of all core components (FSM, AI, ML, IPC) to ensure correctness, stability, and maintainability throughout the development lifecycle.

**🎯 Objectives**

* Deploy a scalable unit testing infrastructure for both Python and C++ components
* Ensure coverage of all protocol logic, AI decisions, FSM transitions, and IPC messaging
* Organize test suites per subsystem for maintainability and clarity
* Integrate automated testing with CI (GitHub Actions) and reporting dashboards
* Establish a baseline for regression testing and future sprint release validation
* Document best practices for writing and interpreting ATROP unit tests

**📦 Epic Deliverables**

* ✅ Python unit test framework using pytest with test discovery and config (T6-01)
* ✅ C++ unit test framework using gtest and CMake targets in daemon modules (T6-02)
* ✅ Per-module test folder layout for AI, ML, protocol, FSM, and IPC (T6-03)
* ✅ Protocol parsing tests for NIV, PIV, IDR, FIF headers (T6-04)
* ✅ FSM state transition tests for valid and invalid flows (T6-05)
* ✅ ML inference validation: vectorization → classification → FIF emission (T6-06)
* ✅ AI decision tests for intent and policy compliance (T6-07)
* ✅ IPC protocol message encoding/decoding tests (T6-08)
* ✅ CI job to auto-trigger tests on push and pull request (T6-09)
* ✅ Test reporting output in HTML/JSON for visibility and debugging (T6-10)
* ✅ Regression test baseline established for stable releases (T6-11)
* ✅ Documentation of testing approach, usage, and maintenance workflows (T6-12)

**🧾 User Story: T6-01 – Setup Python Unit Test Infra**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ Done

**As a** Test Engineer, **I want to** set up the Python unit testing infrastructure using pytest. **So that** Python-based ATROP components can be reliably tested with modular structure, easy discovery, and CI compatibility.

**🎯 Acceptance Criteria**

* pytest installed as the core framework via requirements-dev.txt or pyproject.toml
* Base directory created at test/python/ or tests/
* Module-aligned folder structure established for:
  + ai\_control\_plane/
  + ml\_data\_plane/
  + ipc/
  + fsm/
* Test discovery confirmed with:
  + Standard naming: test\_\*.py
  + Config file: pytest.ini or pyproject.toml section
* Sample dummy test passes under pytest runner
* CI compatibility verified (manual or wired to T6-09)
* Developer guide created: docs/tests/python\_test\_setup.md with how-to-run and add tests

**🧩 Notes**

* Provides test foundation for all Python-driven subsystems (AI engine, IPC, config, feedback)
* Complements C++ test setup (T6-02) and feeds into CI pipelines (T6-09)
* Enables test automation, regression baseline (T6-11), and unit-to-integration test expansion

**🧾 User Story: T6-02 – Setup C++ Unit Test Infra**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ Done

**As a** C++ Developer or Test Engineer, **I want to** establish a unit testing infrastructure using Google Test (gtest) and CMake. **So that** all C++ modules in ATROP (e.g., FSM, IPC, control logic) can be independently tested with automated build integration.

**🎯 Acceptance Criteria**

* Google Test (gtest) added to build system via:
  + External project (FetchContent) or submodule
  + CMake integration in root and daemon/ directories
* Unit test targets created under:
  + daemon/test/ or test/cpp/
  + Organized by module: fsm/, ipc/, config/, logging/
* Each test target:
  + Compiles with make test or cmake --build
  + Links against ATROP C++ modules
  + Executes with standard gtest\_main entry
* Sample test created for:
  + FSM state transition logic
  + Dummy IPC message parse/serialize
* Test discovery wired to CTest or CI runner
* Documented in docs/tests/cpp\_test\_setup.md with build instructions and example structure

**🧩 Notes**

* Required for FSM engine testing (T6-05), protocol parsers (T6-04), and IPC coverage (T6-08)
* Enables full test automation with CI (T6-09) and reporting support (T6-10)
* Forms critical foundation for validation of real-time C++ agent behavior in ATROP daemon stack

**🧾 User Story: T6-03 – Organize Unit Test Layout**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ Done

**As a** QA Lead, **I want to** organize the unit test folder structure per subsystem. **So that** all tests are logically grouped by component (AI, ML, protocol, IPC, FSM), enabling modular testing, easier maintenance, and parallel development.

**🎯 Acceptance Criteria**

* Test directories created under:
  + test/ or tests/ for top-level test organization
* Subdirectories created for each ATROP functional domain:
  + ai\_control\_plane/
  + ml\_data\_plane/
  + protocol/
  + ipc/
  + fsm/
* Both Python (pytest) and C++ (gtest) test files placed under corresponding folders
* Placeholder test files or README.md added in each folder
* Naming conventions documented (e.g., test\_<feature>.py, fsm\_test.cpp)
* Folder layout aligned with CI discovery paths and build system targets
* Documentation added to docs/tests/layout\_overview.md summarizing folder purpose and usage

**🧩 Notes**

* Enables per-module test accountability and extensibility
* Supports future automation, coverage tracking, and sprint-level ownership
* Required for consistent team-wide test structure and contributor onboarding

**🧾 User Story: T6-04 – Write Protocol Parsing Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T2-04 – Parse Protocol Headers  
**Predecessor Status:** 🔵 To Do

**As a** Protocol Developer, **I want to** write unit tests for parsing ATROP protocol headers. **So that** we can validate the accuracy and resilience of the header decoder for NIV, PIV, IDR, and FIF fields under normal and edge conditions.

**🎯 Acceptance Criteria**

* Unit tests written for each ATROP header type:
  + NIV – Node Identity Vector
  + PIV – Path Inference Vector
  + IDR – Intent Descriptor Record
  + FIF – Feedback Injection Format
* Tests include:
  + Well-formed binary or TLV packets with expected field extraction
  + Boundary conditions (min/max field sizes, truncated data)
  + Invalid/malformed header detection and graceful handling
* Format tested matches spec defined in T5-01 (Protobuf or binary schema)
* Implemented in C++ (gtest) or Python (pytest), depending on parsing module language
* Located under test/protocol/ with named test cases per header type
* Documentation added to docs/tests/protocol\_parsing.md with example inputs/outputs

**🧩 Notes**

* Required to ensure protocol safety and interop with external agents or simulators
* Foundation for FSM behavior, feedback processing, and integration flows
* Supports higher-level tests in T7-08 (Packet Capture Verification) and T4-04 (Telemetry Injection)

**🧾 User Story: T6-05 – Write FSM Transition Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T2-02 – Implement FSM Transitions  
**Predecessor Status:** ✅ Done

**As a** Protocol Developer, **I want to** implement unit tests for the ATROP Finite State Machine (FSM). **So that** we can validate correct behavior during valid transitions and ensure proper handling of invalid or unexpected events.

**🎯 Acceptance Criteria**

* Unit tests cover:
  + All valid FSM transitions (e.g., INIT → DISCOVERY → LEARN → DECIDE → ... → EXIT)
  + Edge transitions (e.g., CORRECT → DISCOVERY, FEEDBACK → LEARN)
  + Invalid transitions (e.g., LEARN → INIT) with expected fallback or error handling
* Each test simulates:
  + Initial state, triggering event, and expected next state
  + Optional side effects (e.g., log entries, callback triggers)
* Error handling tested for:
  + Unknown events
  + Malformed state data
  + FSM timeout or stalling scenarios
* Implemented in C++ (gtest) under test/fsm/ with traceable output logs
* FSM definition loaded from config (T1-07) or test fixtures
* Test coverage documented in docs/tests/fsm\_state\_tests.md

**🧩 Notes**

* Validates T2-01 and T2-02 implementation
* Critical for safety, reliability, and autonomous state transitions
* Supports ATZ simulation (T2-09) and FSM correction logic (T2-07) in downstream testing

**🧾 User Story: T6-06 – Write ML Inference Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T4-01 – ML Inference Agent  
**Predecessor Status:** 🔵 To Do

**As an** ML Developer, **I want to** write unit tests for the ML inference agent. **So that** we can validate the flow vectorization process, classification output, and correct triggering of FIF telemetry packets.

**🎯 Acceptance Criteria**

* Tests validate:
  + Flow metadata correctly converted into feature vectors
  + Model returns expected classification label for known input patterns
  + Confidence thresholds and fallback handling logic are exercised
  + FIF telemetry is emitted in response to inference result
* Coverage includes:
  + Normal flows (video, bulk, VoIP)
  + Edge cases (malformed, partial, unknown flows)
  + Resource exhaustion triggers (optional, if safety fallback is active)
* Implemented in Python with pytest under test/ml\_data\_plane/
* FIF payload structure conforms to telemetry spec (T4-04)
* Test logs trace flow → vector → model → label → FIF lifecycle
* Documented in docs/tests/ml\_inference\_tests.md with example vectors and test cases

**🧩 Notes**

* Complements model validation (T4-02), FIF generation (T4-04), and policy orchestration (T3-05)
* Critical for correctness of real-time ML classification and feedback behavior
* Supports full-path test in T4-08 and integration test in T4-10

**🧾 User Story: T6-07 – Write AI Decision Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T3-01 – Build AI Model Engine  
**Predecessor Status:** 🔵 To Do

**As an** AI Developer, **I want to** write unit tests for the AI model engine. **So that** we can ensure its routing decisions align with intent policies, network state, and trust constraints.

**🎯 Acceptance Criteria**

* Tests validate:
  + AI model receives input (e.g., topology state, intents, feedback)
  + Output matches expected routing action or path selection
  + Policies (e.g., latency, zone affinity, bandwidth) are honored
  + Confidence score and fallback behavior evaluated
* Inputs tested include:
  + Static topologies
  + Simulated feedback (FIF, trust degradation)
  + User intent profiles (video, secure, best-effort)
* Coverage includes:
  + Valid policy compliance
  + Policy conflict resolution
  + Path changes due to anomaly triggers
* Implemented in Python with pytest under test/ai\_control\_plane/
* Logs include input → model decision → policy match result
* Documented in docs/tests/ai\_decision\_tests.md with sample input and expected output

**🧩 Notes**

* Verifies T3-01 and T3-05 logic before integration with FSM or control plane
* Supports simulation replay (T3-09) and closed-loop validation in Phase 7
* Ensures AI trustworthiness, policy adherence, and test reproducibility

**🧾 User Story: T6-08 – Write IPC Protocol Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T5-03 – Create IPC Clients  
**Predecessor Status:** 🔵 To Do

**As a** Test Engineer, **I want to** write unit tests for the IPC protocol implementation. **So that** we can validate message encoding, decoding, and connection handling between ATROP agents and the IPC server.

**🎯 Acceptance Criteria**

* Tests verify:
  + Correct serialization and deserialization of all defined IPC message types
  + Message integrity across encoding schemes (e.g., JSON or Protobuf from T5-01)
  + Handling of malformed or truncated messages
  + Connection behavior: connect, disconnect, reconnect
  + Heartbeat and timeout triggers (optional from T5-10)
* Test cases simulate:
  + Message send/receive between mock client and server
  + Agent registration and basic message exchange
* Implemented in Python (pytest) and/or C++ (gtest) depending on client library stack
* Organized under test/ipc/ with input/output validation and schema checks
* Logs trace message lifecycle and error conditions
* Documented in docs/tests/ipc\_protocol\_tests.md with message format examples

**🧩 Notes**

* Validates correctness and reliability of ATROP’s messaging backbone
* Required for integration testbed (T5-09), security validation (T5-07), and config reload testing (T5-06)
* Supports CI-driven message regression detection (T6-09) and onboarding safety for new agents

**🧾 User Story: T6-09 – CI: Auto Trigger Unit Tests**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T1-11 – CI Pipeline Bootstrap  
**Predecessor Status:** ✅ Done

**As a** DevOps Engineer, **I want to** configure GitHub Actions to automatically trigger all unit tests. **So that** every code push or pull request runs Python and C++ test suites, ensuring early detection of issues and consistent code quality.

**🎯 Acceptance Criteria**

* GitHub Actions workflow (.github/workflows/ci.yml) updated to include:
  + Python unit tests (pytest) → test/python/ or tests/
  + C++ unit tests (gtest) → test/cpp/ or daemon/test/
* Triggered on:
  + push to any branch
  + pull\_request to main or dev branches
* Steps include:
  + Dependency installation (pip, cmake, g++)
  + Build test binaries for C++
  + Run all Python tests with coverage
  + Output formatted logs or summaries
* Build matrix supports:
  + Ubuntu latest (default)
  + Optional future extension: multi-platform or Python versions
* CI results visible in PR checks and GitHub Actions tab
* Documented in docs/tests/ci\_pipeline.md with workflow overview and test coverage notes

**🧩 Notes**

* Connects test execution to CI/CD flow, completing dev-test automation loop
* Enables future CI enhancements (e.g., reporting in T6-10, regression tracking in T6-11)
* Ensures test breakage is caught before merge, maintaining code stability across modules

**🧾 User Story: T6-10 – Test Reporting Setup**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T6-01 – Setup Python Unit Test Infra → 🔵 To Do
* T6-02 – Setup C++ Unit Test Infra → 🔵 To Do

**As a** QA Lead, **I want to** enable reporting for test results in HTML and JSON formats. **So that** developers and maintainers have clear visibility into test coverage, failures, and execution trends via automated CI outputs.

**🎯 Acceptance Criteria**

* Python (pytest) reports generated:
  + HTML report using pytest-html
  + JSON summary using --json-report plugin or equivalent
* C++ (gtest) results exported as:
  + XML (--gtest\_output=xml)
  + Optional: Convert to HTML or upload as GitHub artifact
* CI pipeline (T6-09) modified to:
  + Save reports as artifacts
  + Optionally publish coverage badge or summary comment on PR
* Reports include:
  + Total tests, passed/failed/skipped counts
  + Per-module breakdown (AI, ML, FSM, IPC, etc.)
  + Links to logs or coverage insights
* Output structure documented in docs/tests/reporting.md with interpretation guide

**🧩 Notes**

* Critical for tracking progress and catching regressions (T6-11)
* Adds observability to CI workflows and enables future dashboard integration
* Complements unit/integration workflows across all phases and modules

**🧾 User Story: T6-11 – Regression Test Baseline**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T6-01 to T6-08 – All unit test setup and authoring stories  
  **Predecessor Status:** 🔵 To Do

**As a** QA Lead, **I want to** establish a regression test baseline. **So that** known-good test cases can be frozen per sprint or release and used to detect future breakage, ensuring code and model stability over time.

**🎯 Acceptance Criteria**

* Snapshot all passing unit test cases from:
  + FSM (T6-05)
  + Protocol (T6-04)
  + AI/ML (T6-06, T6-07)
  + IPC (T6-08)
* Create a regression/ suite in both Python and C++ test folders
* Mark baseline cases with version tag or commit hash
* CI pipeline updated to:
  + Run regression separately or in parallel with current tests
  + Flag changes in expected output or performance
* Results compared against frozen state using checksum or output diffs
* Process documented in docs/tests/regression\_baseline.md:
  + How to add/remove cases
  + When to update the baseline
  + Review and approval workflow for baseline changes

**🧩 Notes**

* Essential for maintaining behavior consistency across ATROP versions
* Enables confidence during feature addition, refactoring, or model retraining
* Supports long-term test auditability and release stability tracking

**🧾 User Story: T6-12 – Document Testing Strategy**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:** All Phase 6 test setup and execution tasks (T6-01 to T6-11)  
**Predecessor Status:** 🔵 To Do

**As a** Documentation SME, **I want to** document the complete unit testing strategy for ATROP. **So that** contributors, maintainers, and reviewers understand how to write, execute, extend, and interpret tests across all modules and languages.

**🎯 Acceptance Criteria**

* A unified testing strategy document created at:
  + docs/tests/testing\_strategy.md
* Content includes:
  + How to structure new test files for both pytest and gtest
  + Folder layout explanation (ai\_control\_plane/, ipc/, fsm/, etc.)
  + Naming conventions and tagging (e.g., smoke, regression, module-name)
  + How to run tests locally with coverage (pytest, ctest)
  + How to interpret test outputs, CI results, and HTML/JSON reports
  + How to contribute new tests or baseline updates (linked to T6-11)
* Links added to related documentation:
  + ci\_pipeline.md (T6-09)
  + reporting.md (T6-10)
  + regression\_baseline.md (T6-11)
* Reviewed by QA Lead and updated every sprint milestone

**🧩 Notes**

* Essential for maintaining test consistency across contributors and releases
* Helps onboard new engineers and SME testers into ATROP’s verification flow
* Forms the foundation for quality gate enforcement in CI/CD and vendor onboarding

**🧩 Epic: ATROP Phase 7 – Integration Testing & System Behavior Validation**

**📘 Description**

This epic focuses on validating ATROP's full-stack integration through live, emulated, and container-based environments. It ensures that control and data plane components—including AI decisioning, ML classification, FSM transitions, and policy enforcement—interoperate correctly across realistic topologies. These tests span scenarios from intent injection and feedback loops to flow anomaly response and legacy protocol compatibility.

**🎯 Objectives**

* Deploy a containerized or emulated test framework (e.g., Mininet + FRRouting)
* Simulate end-to-end packet and control flow across ATZ zones
* Validate real-time reaction to telemetry, intent, and topology changes
* Confirm cross-agent behavior with policy and FSM enforcement
* Establish repeatable integration regression suite and scenario documentation

**📦 Epic Deliverables**

* ✅ Emulated integration test harness (Mininet or Docker-based ATZ mesh) – T7-01
* ✅ AI/ML feedback-to-decision validation pipeline (flow → FIF → policy → path) – T7-02
* ✅ Intent enforcement test across multiple ATZ nodes – T7-03
* ✅ Topology reactivity test (failure detection, re-learn, re-route) – T7-04
* ✅ Flow behavior classification test (bursty, attack, QoS labeling) – T7-05
* ✅ Multi-zone FSM correction sync scenario – T7-06
* ✅ FRRouting compatibility scenario (OSPF/BGP route handling) – T7-07
* ✅ Header capture and validation (PIV, IDR, FIF structure) – T7-08
* ✅ Frozen test scenarios and CI-regression harness for releases – T7-09
* ✅ Documentation of test topologies, intents, and expected outcomes – T7-10

**🧾 User Story: T7-01 – Integration Test Framework**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T1-04 – Setup Testing Scaffold  
**Predecessor Status:** ✅ Done

**As an** Integration Lead, **I want to** set up a full integration testing harness using Mininet, FRRouting, or a containerized ATZ topology. **So that** we can emulate real-world routing environments and validate ATROP’s end-to-end system behavior across modules and zones.

**🎯 Acceptance Criteria**

* Select and configure testbed platform:
  + Mininet topology (virtual Ethernet mesh)
  + Or Docker-based container emulation (with virtual links)
* Deploy core ATROP agents in container/namespaces:
  + FSM nodes, ML agents, AI controller, IPC daemon
* Include FRRouting (OSPF/BGP) instances for interop testing
* Integrate with CI or local scripts for repeatable launches
* Enable:
  + Traffic injection (e.g., iperf, scapy)
  + Packet capture (e.g., tcpdump, tshark)
  + Fault injection (link down, CPU stress, delay)
* README with topology diagrams, launch instructions, and cleanup steps
* Verified by at least one test: AI + ML feedback path or FSM correction cycle

**🧩 Notes**

* Enables integration stories T7-02 to T7-09
* Required for validating system coordination, protocol compliance, and legacy compatibility
* Forms the foundation for Phases 8 (Simulation) and 10 (Release Staging)

**🧾 User Story: T7-02 – AI + ML Feedback Integration Test**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-04 – Feedback Loop Receiver → 🔵 To Do
* T4-04 – Feedback Injection Generator → 🔵 To Do

**As an** AI/ML Developer, **I want to** simulate a full feedback loop from ML inference to AI route decision. **So that** we can verify that flow behavior triggers telemetry (FIF), which is correctly processed by the control plane to update routing decisions in real time.

**🎯 Acceptance Criteria**

* Emulated ATZ testbed or container setup running:
  + ML inference agent
  + Feedback injection module
  + Feedback receiver (AI control engine)
  + FSM or routing module
* Test simulates live packet flow classified by ML into:
  + High-congestion or threat label
* ML agent generates a FIF packet with trust score + label
* Feedback loop:
  + AI engine receives FIF
  + Policy engine adjusts routing path or applies penalty
  + New route decision sent to FSM or forwarding table
* Logs captured for each stage:
  + Flow → FIF → AI model → Policy → FSM
* Visualized sequence diagram or log timeline for full trace
* Test documented in test/integration/t7\_02\_feedback\_loop/README.md

**🧩 Notes**

* Validates cross-module connectivity and feedback orchestration
* Critical milestone for demonstrating adaptive, policy-aware behavior
* Enables closed-loop trust-aware routing (foundation for Phases 4 and 10)

**🧾 User Story: T7-03 – Intent-Based Routing Validation**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-03 – Intent Processing Unit (IPU) → 🔵 To Do
* T3-05 – Decision Policy Orchestrator → 🔵 To Do

**As a** Policy Tester, **I want to** send intent-based policies to the ATROP control plane. **So that** I can verify that the resulting routing decisions enforce the intended path behavior across Autonomous Topology Zones (ATZ).

**🎯 Acceptance Criteria**

* Emulated ATZ environment deployed with:
  + FSM nodes
  + AI decision engine
  + Intent injection interface
* Sample intents tested:
  + Latency-sensitive (e.g., intent=low-latency)
  + Secure zone preference (e.g., zone=trusted)
  + Bandwidth-heavy bulk transfer
* Routing decisions reflect intent via:
  + Path preference
  + Zone restriction
  + Metric override (trust, bandwidth, policy boost)
* AI decision logs show applied policy and score rationale
* Traffic flows injected and path verified (e.g., traceroute, logs)
* Test documented with:
  + Input intent
  + Observed route
  + Validation outcome
  + Location: test/integration/t7\_03\_intent\_routing/README.md

**🧩 Notes**

* Validates top-down control flow: intent → policy → AI → FSM
* Reinforces ATROP’s core differentiator: autonomous, intent-driven routing
* Complements feedback-based adjustment (T7-02) and policy override (T3-05)

**🧾 User Story: T7-04 – Topology Change Response Test**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-02 – Topology Analytics Engine → 🔵 To Do
* T2-01 – Define Core ATROP States → ✅ Done

**As a** Topology Engineer  
**I want to** simulate node and link failures in an ATZ test environment  
**So that** the AI model can detect topology changes, re-learn the network graph, and update routing decisions accordingly.

**🎯 Acceptance Criteria**

* Deploy testbed with at least 3 interconnected ATZ nodes using:
  + Emulated links (Mininet or container network)
* Topology Analytics Engine (T3-02) actively monitors:
  + Link states
  + Node availability
  + Metrics (latency, loss)
* Simulate failure scenarios:
  + Link down (e.g., tc or ip link set)
  + Node crash (stop container/agent)
* AI model detects change and re-processes:
  + Updated graph structure
  + Trust or performance degradation
* New route computed and applied within acceptable convergence time
* Logs show:
  + Failure detection
  + AI graph update
  + Re-decision event
  + Route injection into FSM or forwarding module
* Test documented in test/integration/t7\_04\_topology\_change/README.md

**🧩 Notes**

* Critical for validating ATROP's adaptive routing capabilities
* Complements intent enforcement (T7-03) and ML feedback loop (T7-02)
* Required before pre-production staging in Phase 8 (T8-04, T8-09)

**🧾 User Story: T7-05 – Flow Behavior Sensitivity Test**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T4-02 – Flow Classification Model → 🔵 To Do
* T4-08 – Flow Behavior Simulation → 🔵 To Do

**As a** Data Plane QA Engineer, **I want to** simulate various traffic patterns (e.g., bursty, video, attack flows). **So that** I can verify that the ML inference engine classifies them correctly and emits the appropriate label and telemetry (FIF).

**🎯 Acceptance Criteria**

* Deploy ATZ node with:
  + Active ML inference agent (T4-01)
  + Configured flow classification model (T4-02)
* Simulate or replay flow types:
  + Bursty TCP/UDP flows
  + Video streaming (e.g., ffmpeg, iperf)
  + Malicious traffic patterns (e.g., SYN flood, slowloris)
* ML agent produces:
  + Correct classification label
  + Associated trust score
  + FIF telemetry with flow vector metadata
* Validate output via:
  + Log capture
  + FIF packet inspection
  + Telemetry debug hooks
* Behavior documented per flow type under:
  + test/integration/t7\_05\_flow\_sensitivity/README.md
  + Expected vs. actual classification and response

**🧩 Notes**

* Validates real-time responsiveness and reliability of ATROP’s ML layer
* Precedes end-to-end trust enforcement and AI adjustment tests
* Required for closing the loop between Phase 4 ML models and Phase 3 policy adaptation

**🧾 User Story: T7-06 – Multi-Zone FSM Sync Test**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T2-01 – Define Core ATROP States → ✅ Done
* T2-05 – Implement ATZ Zone Formation → 🔵 To Do

**As a** Protocol Engineer, **I want to** simulate multiple FSMs operating across different ATZ zones. **So that** I can verify they react appropriately and in sync when correction feedback (e.g., anomalies or trust drops) is issued across zones.

**🎯 Acceptance Criteria**

* Testbed includes at least two distinct ATZ zones with:
  + Independent FSM agents
  + Shared IPC layer for feedback distribution
* Simulate:
  + Discovery → Correction packet sent across zones
  + Anomaly injected in Zone A triggering state update in Zone B
* FSMs respond to:
  + CORRECT event
  + Triggered state fallback or mitigation in peer FSM
* Logs show synchronized or coordinated state transitions
* Event propagation timing and causality validated
* Located under test/integration/t7\_06\_multi\_zone\_fsm/ with:
  + FSM logs
  + Message traces
  + Expected coordination path

**🧩 Notes**

* Ensures distributed FSM logic can coordinate under anomaly conditions
* Required for network-wide convergence and isolation behavior
* Supports T2-07 (correction handler), T2-09 (zone behavior test), and prepares for T8-04 simulation of zone splits

**🧾 User Story: T7-07 – Legacy Protocol Integration Test**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T1-04 – Setup Testing Scaffold → ✅ Done
* Phase 0 – Foundational setup and governance completed → ✅ Done

**As an** FRR Integration Lead, **I want to** connect ATROP nodes to FRRouting daemons running RIP, EIGRP (if supported), OSPF, and BGP. **So that** I can validate interoperability between ATROP’s control plane and standard IGP/EGP routing protocols used in existing networks.

**🎯 Acceptance Criteria**

* Deploy testbed with:
  + One or more ATROP FSM nodes
  + Containers or namespaces running FRRouting with OSPF and BGP enabled
  + Optional: RIP/EIGRP validation if supported by FRR build
* Configure route redistribution:
  + ATROP → OSPF/BGP and vice versa
  + Static intents or learned paths injected into IGP/EGP
* Validate:
  + Bidirectional route visibility
  + Forwarding correctness (ping, traceroute)
  + Loop prevention and filtering logic
* Logs confirm:
  + Proper route origination and receipt
  + Metrics, labels, or policy tags preserved where applicable
* Integration scripts stored in test/integration/t7\_07\_legacy\_frr/
  + Include FRR configs, ATROP route export/import config
  + Test result summary and limitations (e.g., RIP/EIGRP partial support)

**🧩 Notes**

* Ensures backward compatibility with existing infrastructure
* Required for Phase 8 shadow mode (T4-11) and vendor hook validation (T9-10)
* Demonstrates ATROP’s ability to co-exist in hybrid deployments across brownfield and greenfield networks

**🧾 User Story: T7-08 – Packet Capture & Verification**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T2-04 – Parse Protocol Headers → 🔵 To Do
* T4-04 – Feedback Injection Generator → 🔵 To Do

**As a** Packet Analyst, **I want to** capture and inspect live ATROP traffic. **So that** I can verify that encoded packet headers (e.g., PIV, IDR, FIF) conform to protocol specification and carry the correct semantic meaning.

**🎯 Acceptance Criteria**

* Use tcpdump, tshark, or equivalent to capture packets from:
  + ATROP ML agents
  + AI decision nodes
  + FSM/forwarding pipeline
* Extract and analyze the following headers:
  + PIV (Path Inference Vector)
  + IDR (Intent Descriptor Record)
  + FIF (Feedback Injection Format)
  + Optional: NIV (Node Identity Vector)
* Compare captured values to:
  + Input flow metadata
  + Intent/policy definitions
  + Expected output from test scenario
* Verify:
  + Header structure follows binary/TLV spec from T2-04
  + Field values are correctly encoded (IDs, scores, zone ID, etc.)
  + Telemetry (FIF) aligns with flow classification and event timing
* Reports include:
  + Hex and decoded view
  + Field-by-field explanation
  + Screenshot or log from capture tools
* All data stored in test/integration/t7\_08\_packet\_capture/ with:
  + .pcap or .pcapng traces
  + Capture filters
  + Parsing scripts and validation notes

**🧩 Notes**

* Validates protocol correctness, compliance, and field encoding logic
* Complements T6-04 (header parsing tests) and supports T7-02, T7-05
* Required for audit traceability, debugging, and vendor protocol interop analysis

**🧾 User Story: T7-09 – Integration Regression Test Set**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T7-01 to T7-07 – All core integration tests must be implemented  
  **Predecessor Status:** 🔵 To Do

**As a** QA Lead  
**I want to** define and freeze a stable set of integration test scenarios  
**So that** they can be re-used in CI pipelines and during sprint or release validation to detect regressions in system-level behavior.

**🎯 Acceptance Criteria**

* Select and document test scenarios from:
  + T7-02: Feedback loop validation
  + T7-03: Intent-based routing
  + T7-04: Topology failure recovery
  + T7-06: FSM sync across zones
  + T7-07: FRRouting interop
* For each scenario:
  + Define input conditions, configuration, and expected outputs
  + Save all topology and intent files required to run tests
  + Create tagged directory in test/integration/regression/
  + Include automated validation scripts (log parsing, packet checks)
* Tag tests with:
  + Scenario ID (e.g., T7R-01)
  + Phase/version freeze label (e.g., v0.5-baseline)
* Add GitHub Actions or local runner support for periodic CI execution
* Logs stored in artifacts/ or compressed into baseline archive
* Test freeze criteria documented in docs/tests/integration\_baseline.md:
  + When to re-freeze
  + Review and acceptance checklist
  + Expected runtime and CI load

**🧩 Notes**

* Enables stable regression tracking across all future code or model changes
* Supports T10-03 (final validation) and T10-10 (go/no-go decision checklist)
* Anchors ATROP’s integration quality and test reproducibility strategy

**🧾 User Story: T7-10 – Document Integration Scenarios**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T7-01 to T7-05 – Core integration test scenarios  
  **Predecessor Status:** 🔵 To Do

**As a** Documentation SME, **I want to** document all major ATROP integration test scenarios. **So that** developers, testers, and reviewers can understand the testbed topologies, intents, traffic patterns, and expected results used in validation.

**🎯 Acceptance Criteria**

* For each major scenario (T7-01 to T7-05), create a structured documentation entry:
  + scenario\_name.md under docs/tests/integration/
* Each file must include:
  + **Diagram** of testbed topology (PlantUML or embedded PNG)
  + **Intent policy** or configuration used
  + **Flow types** simulated (e.g., bursty, video, bulk, malicious)
  + **Expected behavior** (e.g., route choice, fallback, trust update)
  + **Validation method** (e.g., traceroute, packet capture, AI logs)
  + **Success criteria** and known deviations (if any)
* Optional: Link to recorded .pcap, logs, or CI artifacts
* Index added at docs/tests/integration/README.md listing all scenarios
* All diagrams stored under docs/diagrams/integration/

**🧩 Notes**

* Supports transparency and reproducibility for test and release review
* Provides visual and policy context for T7-09 regression set
* Facilitates onboarding of QA, integration partners, and vendors reviewing ATROP validation flow

**🧩 Epic: ATROP Phase 8 – Emulated Testbed, Simulation & Staging Validation**

**📘 Description**

This epic focuses on building, automating, and leveraging a full emulated ATROP testbed to simulate realistic topologies, stress conditions, protocol redistribution, and real-world packet flows. It culminates in a pre-production staging environment, enabling validation of ATROP’s adaptive routing logic, control/data plane behaviors, and interop capabilities under realistic, fault-prone, or legacy scenarios.

**🎯 Objectives**

* Build and deploy a reusable emulated testbed using Mininet, Docker, or equivalent tools
* Simulate complex topologies (DC, WAN, 5G, metro) using config-driven templates
* Stress the control and data planes through dynamic churn, split/merge events, and policy violations
* Validate legacy compatibility, ML flow analysis, and performance under load
* Capture telemetry, convergence metrics, and prepare for final staging validation
* Document the full framework for ongoing use, regression, and reproducibility

**📦 Epic Deliverables**

* ✅ Emulated ATROP lab with dynamic topology launcher (Mininet/Docker) – T8-01
* ✅ Scenario blueprints for DC, WAN, edge, and metro topologies – T8-02
* ✅ Simulation automation using YAML/JSON-driven configurations – T8-03
* ✅ Zone behavior emulation: ATZ formation, splits, merges, correction response – T8-04
* ✅ Link/node churn, prefix hijack, CPU spike, and failure injection modules – T8-05
* ✅ BGP/OSPF/MPLS/Segment Routing redistribution validation – T8-06
* ✅ Flow replay validation from known PCAPs (e.g., video, anomaly) – T8-07
* ✅ Metrics collection: convergence time, AI vs static paths, path stability – T8-08
* ✅ Full-stack pre-production staging with fallback paths and AI disable modes – T8-09
* ✅ Documentation of tooling, diagrams, traffic classes, metrics, and PCAP sources – T8-10

**🧾 User Story: T8-01 – Build Emulated Testbed**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T7-01 – Integration Test Framework  
**Predecessor Status:** 🔵 To Do

**As a** Platform Engineer, **I want to** build an emulated testbed using Mininet, Docker, or container-based topology tools. **So that** we can simulate ATROP deployments in realistic network environments and validate control/data plane behaviors under varying topologies and conditions.

**🎯 Acceptance Criteria**

* Choose and implement the emulation platform:
  + Mininet with virtual Ethernet topology
  + OR Docker Compose with bridged networks
  + OR hybrid testbed using Linux namespaces and virtual links
* Deploy containers/nodes representing:
  + FSM agents
  + ML inference daemons
  + AI model engine
  + IPC relay or broker
  + FRRouting nodes (optional for interop prep)
* Support:
  + Link latency, jitter, and failure injection
  + Zone isolation and cross-ATZ traffic simulation
  + Basic flow replay (manual or scripted)
* Include basic orchestration script:
  + run\_lab.sh or launch\_emulated\_testbed.py
  + Cleanup/reset script
* Store under tools/testbed/emulation/ with:
  + Topology definitions
  + Startup configs
  + README with architecture and launch steps

**🧩 Notes**

* Required foundation for T8-02 (topology scenarios), T8-09 (staging run)
* Enables system-wide validation in isolation, with reproducibility
* Supports multiple emulation modes (single-host and distributed)

**🧾 User Story: T8-02 – Design Topology Scenarios**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:** T8-01 – Build Emulated Testbed  
**Predecessor Status:** 🔵 To Do

**As a** Testbed Architect, **I want to** design and implement emulated topology scenarios that reflect real-world environments. **So that** we can test ATROP behavior across diverse network architectures like data centers, edge networks, and wide area backbones.

**🎯 Acceptance Criteria**

* Create topology definitions for the following environments:
  + 📦 DC Spine-Leaf (3-tier or Clos fabric)
  + 📡 5G Edge (distributed nodes with leaf/gateway topology)
  + 🌐 WAN Backbone (core mesh with regional aggregation)
  + 🏙️ Metro Ring (circular/dual-homed redundant links)
* For each topology:
  + Define node/link layout
  + Assign roles (FSM, AI, ML, gateway)
  + Inject realistic metrics (latency, bandwidth, trust scores)
* Files stored in:
  + tools/testbed/scenarios/ using YAML, JSON, or Python script format
  + Named clearly, e.g., dc\_spine\_leaf.yaml, 5g\_edge.json
* Each topology validated via:
  + Successful ATROP agent launch
  + Reachability and flow connectivity
  + Visual confirmation (optional: PlantUML or Graphviz diagrams)
* Documented in docs/testbed/scenarios/README.md with:
  + Topology diagram
  + Use case/purpose
  + Known constraints or ideal test cases

**🧩 Notes**

* Enables test reuse in T8-03 (automation), T8-09 (pre-production), and T7 regression flows
* Supports scenario-based validation: anomaly behavior, feedback injection, vendor interop
* Enhances repeatability, coverage, and design fidelity in ATROP testing strategy

**🧾 User Story: T8-03 – Automate Simulation Scripts**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T8-01 – Build Emulated Testbed  
**Predecessor Status:** 🔵 To Do

**As a** Testbed DevOps Engineer, **I want to** automate the testbed simulation process using YAML/JSON-driven configuration scripts. **So that** we can quickly launch, modify, and repeat ATROP topology tests without manual effort.

**🎯 Acceptance Criteria**

* Develop automation tools to read topology definitions (from T8-02) in:
  + YAML or JSON formats
  + With node/link roles, attributes, and metrics
* Implement parser and launcher script:
  + Converts config into runtime container/Mininet setup
  + Brings up ATROP agents per role (FSM, AI, ML, IPC)
  + Applies network shaping (latency, bandwidth, etc.)
* Support CLI flags for:
  + Selecting scenario (--topo dc\_spine\_leaf.yaml)
  + Enabling logging, debug mode, or cleanup
* Create validation utility:
  + Confirms all nodes initialized and reachable
  + Outputs connection matrix or live topology view
* Scripts and configs stored under:
  + tools/testbed/launcher/
  + With examples in examples/automation/
* Documented usage in docs/testbed/automation\_guide.md:
  + Setup, CLI usage, adding new scenarios

**🧩 Notes**

* Enables repeatable integration and staging tests
* Used by CI triggers or manual runners in Phases 7–10
* Reduces onboarding/setup time for developers and QA teams

**🧾 User Story: T8-04 – Simulate ATZ Split/Merge**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T2-05 – Implement ATZ Zone Formation → 🔵 To Do
* T7-06 – Multi-Zone FSM Sync Test → 🔵 To Do

**As a** Protocol Engineer, **I want to** simulate Autonomous Topology Zone (ATZ) splits and merges. **So that** I can observe how FSMs adapt during zone-level convergence events and verify correctness of state coordination and recovery.

**🎯 Acceptance Criteria**

* Use the emulated testbed (from T8-01) with:
  + Multiple ATZs formed via zone ID and FSM boundaries
* Simulate:
  + Zone split: break connectivity between FSM nodes in one ATZ
  + Zone merge: restore link between disjoint FSM clusters
* Validate that:
  + FSMs transition into fallback/correction states appropriately
  + ATZ identities are updated and resynchronized
  + Policy awareness (intent/zone bindings) is preserved or updated
* Log events include:
  + Zone state before/after split or merge
  + FSM reactions (transitions, state changes)
  + Any dropped or redirected traffic
* Store test in test/integration/t8\_04\_atz\_split\_merge/ with:
  + Replay scripts
  + Topology and zone metadata
  + Expected output comparison

**🧩 Notes**

* Validates core zone logic and resilience under dynamic topologies
* Builds on FSM coordination (T2-01, T7-06) and prepares for pre-production validation (T8-09)
* Critical for autonomous behavior in large-scale ATROP deployments

**🧾 User Story: T8-05 – Churn + Failure Injection**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T4-04 – Feedback Injection Generator → 🔵 To Do
* T6-06 – Write ML Inference Tests → 🔵 To Do

**As a** Simulation Engineer, **I want to** simulate real-world churn and failure scenarios in the ATROP emulated testbed. **So that** I can observe how the system reacts to instability, resource stress, and malicious routing conditions.

**🎯 Acceptance Criteria**

* Use emulated topologies (from T8-01/T8-02) to inject:
  + Link flaps (rapid up/down cycles)
  + Node crashes or reboots
  + Traffic blackholes (e.g., iptables -j DROP)
  + Prefix hijack events (false route advertisement)
  + CPU/memory overload on ML or FSM agents
* Trigger and log ML or AI behavior:
  + FIF generation due to instability
  + Model fallback or FSM isolation
  + Trust score degradation
* Validate that:
  + AI engine adjusts routing paths
  + FSMs correctly transition (e.g., ENFORCE → CORRECT)
  + System recovers or isolates problem area
* Include:
  + CLI script to launch failures
  + Output logs and packet captures
  + Comparison of expected vs. actual behavior
* Test stored in test/integration/t8\_05\_churn\_failures/
  + With attack vector definitions and mitigation notes

**🧩 Notes**

* Critical to prove ATROP’s fault tolerance and anomaly response
* Supports development of confidence models and safety logic (T3-06, T4-06)
* Required for staging and release signoff (T8-09, T10-10)

**🧾 User Story: T8-06 – Legacy Redistrib. Validation**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T7-07 – Legacy Protocol Integration Test  
**Predecessor Status:** 🔵 To Do

**As a** Legacy Interop SME, **I want to** validate route redistribution between ATROP and traditional protocols like RIP, EIGRP, BGP, OSPF, MPLS, and Segment Routing. **So that** we ensure ATROP can interoperate with brownfield environments and hybrid networks during transition phases.

**🎯 Acceptance Criteria**

* Set up FRRouting or equivalent with support for:
  + RIP (if supported)
  + EIGRP (limited, optional)
  + OSPF
  + BGP
  + Segment Routing (via MPLS simulation or label-based path injection)
* Configure ATROP agents to:
  + Redistribute received routes into legacy protocols
  + Accept and tag external routes with trust/policy indicators
  + Maintain consistent forwarding tables without loops
* Simulate routing exchange scenarios:
  + BGP-to-ATROP route injection
  + OSPF area boundary redistribution
  + Segment label handling in forwarding plane
  + Path preference overrides and policy enforcement
* Validate via:
  + Route tables and logs
  + Packet forwarding accuracy
  + Protocol-specific debug output
* Document results in test/integration/t8\_06\_legacy\_redistribution/ with:
  + Configs for FRR and ATROP agents
  + Redistribution rules and expected behavior
  + Coverage map (protocols, route types, filtering)

**🧩 Notes**

* Ensures full legacy interoperability as mandated by ATROP protocol goals
* Critical for vendor testing, RFC alignment, and phased deployment adoption
* Complements T7-07 and prepares for T10-09 (final vendor hook validation)

**🧾 User Story: T8-07 – Flow Replay Testing**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T4-08 – Flow Behavior Simulation → 🔵 To Do
* T6-06 – Write ML Inference Tests → 🔵 To Do

**As an** ML QA Engineer, **I want to** replay known PCAP traffic traces (e.g., video streaming, bursty flows, malicious patterns). **So that** I can verify the ATROP ML inference engine accurately classifies flow behavior and generates correct trust scores and FIF telemetry.

**🎯 Acceptance Criteria**

* Select and curate PCAPs covering:
  + 📺 Video streaming (e.g., MPEG-DASH, RTP)
  + 🔄 Bursty or periodic flows (e.g., RPC, sync)
  + 🚨 Attack traffic (e.g., port scans, DDoS, crafted payloads)
* Replay traffic using:
  + tcpreplay, scapy, or replay.py against ML inference daemon
* ML engine must:
  + Extract flow vectors
  + Classify with correct label and confidence
  + Emit FIF packets if thresholds are breached
* Validate output by:
  + Checking classification logs
  + Verifying telemetry content (PIV, FIF, trust score)
  + Comparing output against expected labels per PCAP
* Store test scripts, flows, and expectations in:
  + test/integration/t8\_07\_flow\_replay/
  + Include config files, classification baseline, FIF logs
* Document test coverage, PCAP sources, and outcomes in a summary markdown

**🧩 Notes**

* Validates ML behavior under real-world conditions
* Bridges synthetic tests (T6-06, T7-05) and dynamic emulation (T8-01)
* Ensures replayable, deterministic verification before production rollout (T8-09)

**🧾 User Story: T8-08 – Capture Metrics & Logs**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-05 – Decision Policy Orchestrator → 🔵 To Do
* T7-09 – Integration Regression Test Set → 🔵 To Do

**As an** Observability Engineer, **I want to** capture performance and behavior metrics from ATROP under test. **So that** we can evaluate convergence time, path changes, and compare AI-based vs. static routing behaviors across scenarios.

**🎯 Acceptance Criteria**

* Instrument integration test scenarios (from T7-09) to log:
  + 🕒 Convergence time (from event to final route)
  + 🔁 Path change frequency per ATZ node
  + 🧠 AI vs 🛣️ static route selection paths (and divergence count)
  + 📉 Trust scores, route stability scores, fallback activations
* Implement logging endpoints or parsers for:
  + AI model decisions (T3-05)
  + FSM transitions
  + Forwarding decisions and route tables
* Store logs in:
  + logs/metrics/ with scenario and timestamp tags
  + CSV/JSON format for analysis
  + Optional: plot convergence time over multiple runs
* Integrate into CI metrics post-processing (optional for T6-10)
* Document metrics types, units, and logging endpoints in:
  + docs/observability/metrics\_reference.md
  + Include data structure examples and sampling frequency

**🧩 Notes**

* Provides measurable KPIs for ATROP decision intelligence
* Required for performance benchmarking, vendor reports, and release gating (T10-04, T10-10)
* Supports comparison between ML-only, hybrid, and fallback behaviors

**🧾 User Story: T8-09 – Pre-Production Staging Run**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessors:** All prior emulation, integration, feedback, and failure testing stories (T8-01 to T8-08)  
**Predecessor Status:** 🔵 To Do

**As a** Systems QA Lead, **I want to** execute a full-stack ATROP test within a realistic staging environment. **So that** we can validate system-wide interoperability, failover readiness, and release stability prior to production rollout or vendor delivery.

**🎯 Acceptance Criteria**

* Deploy a complete ATROP stack in an emulated lab environment:
  + ML inference agents
  + AI decision model engine
  + FSM agents across multiple ATZ zones
  + IPC backbone
  + Legacy protocol interop (OSPF/BGP/MPLS/SR)
  + Optional: FRRouting or real control-plane peers
* Execute integrated test suite including:
  + Flow classification + FIF feedback → AI → FSM path enforcement
  + Intent policy injection and resolution
  + Topology churn, failure, and correction fallback
  + Legacy redistribution and hybrid routing scenario
* Enable fallback protocols:
  + ATROP → default to static or OSPF route upon ML/AI failure
  + Validate transition time and correctness
* Collect:
  + Logs, metrics (from T8-08)
  + Test results and regressions (from T7-09)
  + Visual FSM and route state transitions
* Store results under:
  + test/staging/t8\_09\_preproduction\_run/
  + Including: topology file, logs, PCAPs, test matrix
* Summary report with:
  + Pass/fail per feature
  + Observations on resilience, correctness, and timing
  + Recommendations for go/no-go criteria (used in T10-10)

**🧩 Notes**

* Simulates full production workload and integration complexity
* Precursor to formal signoff, release packaging, and vendor evaluation
* Ensures deterministic behavior across fallback, AI/ML sync, and policy flows

**🧾 User Story: T8-10 – Document Simulation Framework**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T8-01 to T8-09 – All simulation setup, execution, and validation tasks  
  **Predecessor Status:** 🔵 To Do

**As a** Documentation SME, **I want to** document the entire ATROP simulation framework. **So that** developers, testers, vendors, and external reviewers can understand how to deploy, run, and evaluate the testbed and simulation suite.

**🎯 Acceptance Criteria**

* Create a centralized document at:
  + docs/simulation/README.md
* For each scenario (T8-02 through T8-09), document:
  + ✅ Purpose and scope
  + 🗺️ Topology diagram (or link to image)
  + 🔧 Toolchains used (e.g., Docker, Mininet, tcpreplay)
  + 📦 Files and config paths (e.g., tools/testbed/scenarios/)
  + 🧪 Metrics collected and logs expected (from T8-08)
  + 📁 PCAPs used or generated (replay source or traffic logs)
* Link to:
  + Automation scripts (T8-03)
  + Flow replay tests (T8-07)
  + Metrics dashboards or sample logs
  + Known issues or limitations
* Add a visual index of supported testbed scenarios with:
  + Icons, names, use cases (e.g., DC, WAN, metro, anomaly)
  + Status: implemented / tested / staging
* Validate structure, internal links, and example commands

**🧩 Notes**

* Final reference before staging and release checklist execution (T10-10)
* Helps external contributors replicate test results
* Required for vendor onboarding and future ATROP test suite expansion

**🧩 Epic: ATROP Phase 9 – API Contract, SDKs & External Integration**

**📘 Description**

This epic covers the definition, implementation, and validation of ATROP’s external API interfaces for control, telemetry, and model interaction. It includes the development of SDKs for Python and C++, secure API design, YANG modeling for network integration, and vendor-specific hook implementation. This enables automation, observability, and integration with 3rd-party systems, controllers, and NMS/OSS platforms.

**🎯 Objectives**

* Define the complete API contract for all control, telemetry, and model lifecycle operations
* Deliver Python and C++ SDKs with CLI, REST, and gRPC bindings
* Enable external systems to push policies, monitor state, upload models, and read trust metrics
* Implement security controls (role-based access, certs, tokens)
* Integrate with vendor systems like JunOS, IOS-XR, and NETCONF
* Provide full documentation, usage guides, and integration examples

**📦 Epic Deliverables**

* ✅ API contract covering:
  + /inject, /telemetry, /models, /zones, /config, /feedback (T9-01)
* ✅ Python SDK:
  + Auth, CLI tools, REST/gRPC bindings (T9-02)
* ✅ gRPC definitions (.proto):
  + Real-time telemetry, control messages, model operations (T9-03)
* ✅ C++ SDK wrappers:
  + Edge agent embedding and static API access (T9-04)
* ✅ Telemetry API:
  + Export FSM state, trust metrics, path logs (T9-05)
* ✅ Model upload endpoint:
  + Runtime updates of AI/ML agents (T9-06)
* ✅ YANG model definitions:
  + For ATROP telemetry + configuration (RFC 6020+ compliance) (T9-07)
* ✅ Secure API Layer:
  + Role-based access, tokens, cert authentication (T9-08)
* ✅ SDK Unit Tests:
  + Schema validation, call structure, error handling (T9-09)
* ✅ Sample Vendor Hooks:
  + Integration with JunOS, IOS-XR, EOS (T9-10)
* ✅ SDK Usage Docs:
  + Examples, flows, diagrams, CLI/GitHub-ready (T9-11)

**🧾 User Story: T9-01 – Define API Contract**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** Phase 0 foundational tasks → ✅ Done

**As an** API Architect  
**I want to** define a structured and extensible API contract for the ATROP control and data plane interfaces  
**So that** external systems, SDKs, and vendor integrations can interact with ATROP securely and consistently.

**🎯 Acceptance Criteria**

* Define all core REST and gRPC endpoints, grouped by function:
  + /inject – Inject user intents, policy updates
  + /telemetry – Pull path decisions, FSM state, trust scores
  + /models – Upload, version, or rollback AI/ML models
  + /zones – Get ATZ topology, zone IDs, and FSM mappings
  + /config – Update runtime parameters
  + /feedback – Push telemetry feedback (e.g., FIF)
* For each endpoint, document:
  + HTTP method or RPC call
  + Request and response schema (JSON or .proto)
  + Auth requirements
  + Example payloads and error codes
* Deliverables:
  + api/openapi.yaml for REST schema
  + api/atrop.proto for gRPC services
  + Markdown summary in docs/api/contract.md
* Include extension strategy for future fields/paths
* Validate contract internally with test mocks or Postman collection

**🧩 Notes**

* Enables downstream tasks: SDKs (T9-02, T9-04), telemetry (T9-05), security (T9-08)
* Forms baseline for external integrations and documentation (T9-10, T9-11)
* Should align with naming, versioning, and backward compatibility goals of ATROP API platform

**🧾 User Story: T9-02 – Python SDK Core**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** Python Developer, **I want to** implement a Python SDK for ATROP with authentication, CLI tools, and full REST/gRPC API bindings. **So that** users and systems can easily interact with ATROP services programmatically and from the command line.

**🎯 Acceptance Criteria**

* Implement atrop\_sdk/ package with:
  + Authentication wrapper (API key, token, optional cert)
  + HTTP client for REST endpoints defined in T9-01
  + gRPC stub clients generated from atrop.proto
* Support the following operations via SDK methods:
  + inject\_intent()
  + get\_telemetry()
  + upload\_model()
  + list\_zones()
  + push\_feedback()
* Add CLI tool (atropctl) using argparse or click with subcommands:
  + atropctl inject --file intent.json
  + atropctl telemetry --zone z1
  + atropctl model upload model\_v2.onnx
* Provide:
  + Installation via setup.py or pyproject.toml
  + API documentation using docstrings + Sphinx/Markdown
  + Example scripts in examples/sdk/
* Code must be:
  + Unit-tested (for T9-09)
  + Lint-compliant (flake8/pylint)
  + Compatible with Python 3.8+

**🧩 Notes**

* Enables automation, debugging, and orchestration around ATROP agents
* Forms reference SDK for future language ports (e.g., T9-04 C++)
* Critical for operationalizing control/telemetry flows and validating API readiness

**🧾 User Story: T9-03 – gRPC API Definition**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** gRPC Engineer, **I want to** define .proto files for ATROP’s real-time telemetry and policy injection services. **So that** clients can use high-performance, strongly typed, bidirectional communication channels with ATROP components.

**🎯 Acceptance Criteria**

* Create proto/atrop.proto file containing:
  + InjectIntentService with InjectRequest / InjectResponse
  + TelemetryService with ZoneTelemetryRequest / ZoneTelemetryResponse
  + ModelUpdateService for model lifecycle (upload, list, rollback)
  + FeedbackStream service using stream for FIF packets
* Define messages for:
  + Intent object schema
  + Zone state view
  + AI/ML model metadata
  + Trust scores and feedback telemetry
* Follow gRPC design best practices:
  + Use google.protobuf.Timestamp, Any, and Struct as needed
  + Version service (e.g., rpc GetTelemetryV1)
  + Allow forward compatibility and field extensions
* Generate Python stubs and validate build:
  + python -m grpc\_tools.protoc with output to atrop\_sdk/grpc/
* Add unit test stubs for all gRPC service calls
* Document .proto layout in docs/api/grpc\_reference.md

**🧩 Notes**

* Enables high-performance telemetry and command integration
* Forms binary contract for both SDK (T9-02) and C++ wrappers (T9-04)
* Ensures interoperability across agents, UI, CI pipelines, and external orchestrators

**🧾 User Story: T9-04 – C++ SDK Stubs**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** C++ Developer, **I want to** create C++ SDK stubs for ATROP's control and telemetry APIs. **So that** embedded systems (e.g., edge routers or sensors) can interact with ATROP using lightweight, statically compiled client libraries.

**🎯 Acceptance Criteria**

* Implement a minimal C++ client SDK with wrappers for:
  + REST API calls using libcurl or similar
  + gRPC calls using grpc++ generated from atrop.proto (T9-03)
* Provide function signatures for:
  + bool InjectIntent(const std::string& json\_payload);
  + TelemetryResponse GetTelemetry(const std::string& zone\_id);
  + bool UploadModel(const std::string& filepath);
  + FeedbackStatus PushFIF(const FeedbackMessage& msg);
* Deliverables:
  + Header files: include/atrop\_sdk/atrop\_client.hpp
  + Source files: src/atrop\_sdk/
  + Build system: CMakeLists.txt with gRPC and Protobuf dependencies
* Add sample usage:
  + examples/cpp/inject\_intent.cpp
  + Static linking option for embedded builds
* Validate basic gRPC call connectivity in a testbed

**🧩 Notes**

* Enables real-time participation of embedded/low-footprint devices in ATROP ecosystem
* Complements Python SDK (T9-02) and gRPC definitions (T9-03)
* Used in deployments where Python runtime is unavailable (e.g., industrial routers, edge platforms)

**🧾 User Story: T9-05 – Telemetry API Implementation**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-05 – Decision Policy Orchestrator → 🔵 To Do
* T4-04 – Feedback Injection Generator → 🔵 To Do

**As a** Protocol Developer, **I want to** implement telemetry API endpoints that expose path decisions, ATZ trust metrics, and FSM state logs. **So that** external systems can monitor ATROP's real-time behavior, validate routing intelligence, and audit decision history.

**🎯 Acceptance Criteria**

* Implement the following REST and/or gRPC API endpoints as defined in T9-01:
  + GET /telemetry/paths – current path decisions per flow or zone
  + GET /telemetry/trust – trust scores and their evolution per ATZ
  + GET /telemetry/fsm – recent FSM transitions and active states
* Source data from:
  + AI decision outputs (T3-05)
  + Trust computation modules
  + FSM state engine logs
* Ensure:
  + JSON or protobuf response format
  + Timestamped entries
  + Filter support (?zone=atz-2, ?flow\_id=xyz)
* Write integration tests that:
  + Simulate telemetry access under test conditions
  + Validate schema, completeness, and response latency
* Document API behavior in:
  + docs/api/telemetry.md
  + Include examples, expected payloads, and field descriptions

**🧩 Notes**

* Critical for observability, analytics, and automated policy auditing
* Enables real-time dashboarding, alerting, and system health monitoring
* Supports full-stack visibility during integration (T7), staging (T8), and release phases (T10)

**🧾 User Story: T9-06 – Model Upload Endpoint**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-01 – Build AI Model Engine → 🔵 To Do
* T4-01 – ML Inference Agent → 🔵 To Do

**As an** ML Platform Developer, **I want to** implement an API endpoint that allows uploading and versioning new AI/ML models. **So that** updated inference and control logic can be deployed into the ATROP control and data planes without downtime.

**🎯 Acceptance Criteria**

* Implement REST or gRPC endpoint:
  + POST /models/upload
  + Accepts model file (e.g., .onnx, .pt, .pb)
  + Accepts metadata (version, zone target, model type, checksum)
* Store models under versioned directories (e.g., models/v2.1/)
* Notify relevant modules (AI engine or ML inference agent) to reload model
  + Trigger internal IPC reload or config watcher
* Validate:
  + File integrity (SHA256 or size check)
  + Model compatibility with zone/type
* Handle errors: invalid format, unsupported model, upload failure
* Document usage:
  + Example curl or SDK usage
  + Expected schema and response
  + Lifecycle behavior
* Store test scripts in test/integration/t9\_06\_model\_upload/
  + Include test models and mock updates

**🧩 Notes**

* Enables safe and dynamic control over AI/ML logic without restarts
* Supports federated learning pipelines and per-zone fine-tuning
* Essential for CI/CD of intelligence layers and model rollbacks (future T3-08)

**🧾 User Story: T9-07 – YANG Model Generation**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** NETCONF Engineer, **I want to** define YANG models representing ATROP’s control and telemetry structures. **So that** ATROP can be managed using standards-based network management protocols like NETCONF/RESTCONF in vendor and enterprise environments.

**🎯 Acceptance Criteria**

* Define a YANG module:
  + atrop-telemetry.yang
  + atrop-policy.yang
  + atrop-model-management.yang
* Cover key structures from T9-01:
  + Path decisions
  + ATZ zones and FSM state
  + Trust metrics
  + Intent policy schema
  + Model versioning info
* Follow YANG standards:
  + RFC 6020 / RFC 7950 compliant
  + Include namespace, revision date, organization
  + Use container, leaf, list, typedef, grouping appropriately
* Validate models using:
  + pyang and yanglint
  + At least one simulated get-config/get operation
* Store files under:
  + models/yang/
  + Add README with loading instructions
* Document usage in docs/integration/netconf\_yang.md
  + Example RPCs, data trees, capabilities advertisement

**🧩 Notes**

* Required for interoperability with SDN/NMS platforms and legacy orchestration
* Enables intent mapping and telemetry exposure through standard controllers
* Supports vendor certification and MIB-like registration in enterprise networks

**🧾 User Story: T9-08 – Secure API Layer**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** Security Engineer, **I want to** enforce authentication and access control for all ATROP API endpoints. **So that** only authorized clients and users can interact with control, telemetry, and model operations in a secure and auditable manner.

**🎯 Acceptance Criteria**

* Implement authentication mechanisms:
  + ✅ API token (Bearer token via headers)
  + ✅ Optional X.509 client certificate validation
  + ✅ Optional JWT support for third-party orchestration
* Implement role-based access control (RBAC):
  + admin: full access to all APIs
  + observer: read-only access to telemetry and FSM logs
  + model-uploader: restricted to model lifecycle operations
  + intent-operator: allowed to inject policies/intents
* Configure authorization middleware for:
  + REST endpoints defined in T9-01
  + gRPC services via interceptor hooks
* Validate:
  + Unauthorized requests are rejected with proper error codes
  + Role-enforced logic prevents privilege escalation
* Provide configuration:
  + User/role/token registry via JSON/YAML
  + Environment variables for cert paths and token secrets
* Document in docs/security/api\_access.md:
  + Token issuance
  + Role scopes and permissions
  + TLS setup with cert-based mTLS

**🧩 Notes**

* Enables secure integration in vendor, enterprise, and cloud environments
* Required for T9-10 (vendor hooks) and pre-production staging (T8-09, T10-10)
* Ensures ATROP APIs remain protected from unauthorized injection or observation

**🧾 User Story: T9-09 – Unit Tests for SDK**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessors:**

* T9-02 – Python SDK Core → 🔵 To Do
* T9-03 – gRPC API Definition → 🔵 To Do
* T9-04 – C++ SDK Stubs → 🔵 To Do

**As an** SDK QA Engineer, **I want to** develop unit tests for the Python and C++ ATROP SDKs. **So that** all client methods, schemas, and error-handling logic are verified in isolation before integration.

**🎯 Acceptance Criteria**

* For Python SDK (T9-02):
  + Write pytest-based tests for each public method
  + Use unittest.mock to stub API/gRPC responses
  + Validate schema generation, payload serialization, and CLI tool behavior
  + Ensure proper exceptions are raised on invalid input, auth failures, and timeouts
* For C++ SDK (T9-04):
  + Write gtest cases for REST/gRPC wrapper methods
  + Validate struct parsing, response decoding, error handling
  + Include negative tests for invalid credentials, malformed requests
* Achieve ≥ 85% code coverage for both SDKs
* Store test files in:
  + test/sdk/python/
  + test/sdk/cpp/
* Integrate test jobs into GitHub CI workflow:
  + Triggered on push or PR to sdk/
* Document SDK test instructions in docs/dev/sdk\_testing.md

**🧩 Notes**

* Required for T6-11 (regression baseline) and T10-10 (final release checklist)
* Ensures SDK readiness for external developers and automation tooling
* Enables early detection of breaking changes to API contract or integration flows

**🧾 User Story: T9-10 – Vendor Hook Integration**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:** T9-01 – Define API Contract  
**Predecessor Status:** 🔵 To Do

**As a** Vendor Integration Lead, **I want to** implement sample integration hooks for network vendor platforms (e.g., JunOS, IOS-XR, EOS). **So that** ATROP can interoperate with existing router OS and control-plane environments in brownfield or hybrid deployments.

**🎯 Acceptance Criteria**

* Identify integration targets:
  + Juniper (e.g., JunOS event-policy, op-script, YANG)
  + Cisco (e.g., IOS-XR EEM, YANG, telemetry parser)
  + Arista EOS (e.g., eAPI or CVP hooks)
* For each platform, implement:
  + Sample script or hook to extract routing state or policy feedback
  + Connector that maps to ATROP API endpoints (T9-01)
  + Optional: policy push from ATROP to vendor via CLI/API
* Package example configurations/scripts for:
  + Passive monitoring (read-only, e.g., BGP routes, SNMP, telemetry)
  + Active control (intent injection, path redirection, trust-tag translation)
* Provide vendor-specific readmes:
  + integration/junos/README.md
  + integration/ios-xr/README.md
* Validate in lab/staging testbed:
  + End-to-end telemetry or config exchange
  + Stability, security (auth), and error handling

**🧩 Notes**

* Critical for real-world vendor adoption and RFC-aligned interop
* Supports brownfield deployments and intent bridging across protocols
* Used in staging (T8-09), legacy validation (T8-06), and final sign-off (T10-09)
* Helps demonstrate extensibility to Cisco, Juniper, and open hardware partners

**🧾 User Story: T9-11 – Document SDK Usage**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T9-01 to T9-09 – All API, SDK, and testing components → 🔵 To Do

**As a** Documentation SME, **I want to** create user-friendly documentation for the ATROP SDKs. **So that** developers, integrators, and vendors can easily consume, test, and extend ATROP APIs using provided SDK tools.

**🎯 Acceptance Criteria**

* Write SDK usage guides in:
  + docs/sdk/python\_usage.md
  + docs/sdk/cpp\_usage.md
* Include examples for:
  + Authentication and environment setup
  + Injecting intents and policies
  + Fetching telemetry and FSM state
  + Uploading and rolling back models
  + Using CLI tools (atropctl) with real JSON input
* Document:
  + gRPC service usage via auto-generated Python stubs
  + C++ SDK class references and integration guidance
  + API response structures and common errors
* Add visuals:
  + SDK architecture diagram
  + Flowchart: user → SDK → ATROP agent → result
  + Screenshot or terminal examples
* Ensure docs are:
  + Linked from README.md
  + Verified against real SDK calls or unit tests (T9-09)

**🧩 Notes**

* Final milestone to complete Epic 9 and unblock downstream integration and vendor adoption (T9-10)
* Required for onboarding external contributors and CI automation consumers
* Should match OpenAPI and .proto contract details exactly for consistency

**🧩 Epic: ATROP Phase 10 – Packaging, Release & Production Readiness**

**📘 Description**

This epic delivers the final packaging, system service setup, validation, and deployment documentation necessary for ATROP to be production-ready. It includes building DEB/RPM/Container bundles, creating service units, validating the release over golden topologies, and profiling its performance. The epic also covers fault tolerance, licensing/security audits, and formal release sign-off.

**🎯 Objectives**

* Package ATROP components into standardized, deployable formats
* Validate across full-stack topologies, failure cases, and vendor hooks
* Enable system services with robust logging, restart, and observability
* Provide a reproducible release candidate and deployment runbook
* Complete final performance benchmarks and audit compliance
* Deliver the formal go/no-go checklist with sign-off gating

**📦 Epic Deliverables**

* ✅ DEB/RPM/Container packaging for daemon, AI engine, and ML agent (T10-01)
* ✅ Systemd service units (atrop-agent.service, atrop-ctrl.service) and config templates (T10-02)
* ✅ Final test run across validated golden topologies: WAN, MPLS, DC, 5G (T10-03)
* ✅ Performance profiling for CPU, memory, latency, and inference time (T10-04)
* ✅ Fault isolation hooks for auto-restart, safe-mode fallback (T10-05)
* ✅ SBOM, CVE scan, and license verification audit (T10-06)
* ✅ Reproducible release snapshot: source, binary, test, doc bundle (T10-07)
* ✅ Deployment runbook for production/field engineers (T10-08)
* ✅ Final validation of vendor integrations (e.g., JunOS, EOS) (T10-09)
* ✅ Go/No-Go checklist including test coverage, CI results, regression, and vendor pass (T10-10)

**🧾 User Story: T10-01 – Package Agent Stack**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessors:**

* T1-01 – Scaffold Build System → ✅ Done
* T5-10 – Daemon Identity and Uptime Tracker → 🔵 To Do

**As a** Release Engineer, **I want to** bundle the ATROP daemons into DEB, RPM, and container formats. **So that** the system can be easily deployed, versioned, and maintained across diverse environments (Linux servers, containers, network appliances).

**🎯 Acceptance Criteria**

* Package the following components:
  + atrop-daemon (FSM/AI core engine)
  + atrop-agent (ML inference component)
  + Config files and logging hooks
* Build outputs:
  + .deb (Debian/Ubuntu)
  + .rpm (RHEL/CentOS)
  + Docker/OCI container image with version tag
* Package metadata must include:
  + Name, version, description, maintainer
  + Dependency declarations
  + Post-install script to enable services (T10-02)
* Versioning must align with Git tags or VERSION file:
  + Semantic versioning (e.g., v1.0.0-beta)
  + Snapshot tag support for CI builds
* Validate packages:
  + Install, start, remove, and update across Ubuntu and CentOS
  + Run basic smoke test (validate daemon startup)
* Store output artifacts under:
  + dist/packages/
  + docker/ folder for container definition
* Document packaging process in docs/release/package\_building.md

**🧩 Notes**

* Required for final deployment readiness and vendor distribution (T10-09)
* Enables staging validation (T8-09) and systemd integration (T10-02)
* Must pass through license and security audits (T10-06) before formal release snapshot (T10-07)

**🧾 User Story: T10-02 – Create Systemd Service Files**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:**

* T5-10 – Daemon Identity and Uptime Tracker → 🔵 To Do

**As a** Platform Developer, **I want to** create systemd unit files for ATROP agents and controllers. **So that** the services can be managed, auto-started, and monitored by the host operating system in production deployments.

**🎯 Acceptance Criteria**

* Create systemd unit files:
  + atrop-agent.service – ML inference + feedback handler
  + atrop-ctrl.service – AI control plane daemon + FSM logic
* Include configuration template file (/etc/atrop/atrop.conf):
  + Path to model directory, log directory, IPC socket
  + Zone ID and role (agent/controller)
* Unit file requirements:
  + Restart=on-failure
  + ExecStart=/usr/bin/atrop-agent with config
  + Logging via StandardOutput=journal and SyslogIdentifier=atrop
  + EnvironmentFile=/etc/atrop/atrop.env for runtime overrides
* Validate:
  + Start/stop/restart behavior via systemctl
  + Service registration post-install from DEB/RPM (T10-01)
  + Recovery after crash or reboot
* Include service installation commands in deployment docs:
  + docs/deployment/systemd.md
  + With systemd enable/start guidance

**🧩 Notes**

* Ensures ATROP operates as a native OS-managed service
* Required for packaging completion (T10-01), deployment runbook (T10-08), and production automation
* Contributes to fault isolation and uptime guarantees (T10-05)

**🧾 User Story: T10-03 – Final Golden Topology Validation**

**Story Points:** **5**  
**Status:** 🔵 To Do  
**Predecessor:**

* T8-09 – Pre-Production Staging Run → 🔵 To Do

**As a** QA Lead, **I want to** execute all ATROP test cases across final validated topologies (DC, WAN, MPLS). **So that** we can confirm full system functionality, resilience, and performance before release.

**🎯 Acceptance Criteria**

* Deploy final golden topologies in emulated or containerized testbed:
  + Data Center (spine-leaf, east-west)
  + WAN Backbone (multi-hop, multi-zone, MPLS)
  + 5G/Metro Edge with trust/fallback zones
* Execute:
  + All FSM + AI/ML integration tests
  + Topology churn and failover recovery
  + Intent policy and zone enforcement
  + Feedback loop correction and learning convergence
  + Legacy protocol redistribution (BGP, OSPF, etc.)
* Collect and review:
  + Packet captures, state transitions, telemetry logs
  + Path decisions vs. expected outcomes
  + ML/AI confidence scoring behavior
* Log results:
  + test/final\_topology/ with scenario ID, test results, metrics
* Define pass criteria:
  + 100% critical test pass
  + Acceptable performance thresholds met
  + Zero crash or isolation failure
* Summarize in validation report:
  + Success/failure matrix
  + Observed issues + mitigation
  + Readiness status for T10-07 and T10-10

**🧩 Notes**

* Final blocking step before release sign-off
* Must validate all major flows: control, telemetry, inference, feedback, fallback
* Output feeds into formal release snapshot and go/no-go checklist

**🧾 User Story: T10-04 – Performance Profiling**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T3-01 – Build AI Model Engine → 🔵 To Do
* T4-01 – ML Inference Agent → 🔵 To Do

**As a** Performance Analyst, **I want to** benchmark ATROP’s runtime performance across control and data planes. **So that** we can measure system efficiency, detect bottlenecks, and ensure real-time responsiveness under production conditions.

**🎯 Acceptance Criteria**

* Measure key performance metrics:
  + CPU usage per daemon (AI engine, FSM, ML agent)
  + Memory footprint in active and idle state
  + ML inference time per packet/flow vector
  + End-to-end packet latency (input → decision → action)
  + Zone convergence time under churn
* Use appropriate tools:
  + perf, htop, psrecord, valgrind, bpftrace, time, tc, iperf
  + Custom timers for FSM + model inference instrumentation
* Validate on 3 environments:
  + Containerized testbed
  + VM-based simulation
  + Bare-metal (if available)
* Store profiling results in:
  + perf/summary.md
  + perf/raw/ CSV/JSON snapshots
* Identify:
  + Any anomalies, regressions, or scalability thresholds
  + Minimum CPU/memory spec for edge nodes
* Tag versions used during test:
  + Daemon and model versions, test topology, workload scenario

**🧩 Notes**

* Required for deployment sizing, hardware compatibility, and vendor SLAs
* Performance regressions must be remediated before release (T10-07, T10-10)
* Informs fallback hooks (T10-05) and production runbook constraints (T10-08)

**🧾 User Story: T10-05 – Fault Isolation Hooks**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessors:**

* T5-02 – Build IPC Server (Daemon Hub) → 🔵 To Do
* T6-06 – Write ML Inference Tests → 🔵 To Do

**As a** Platform Engineer, **I want to** implement fault isolation and recovery mechanisms within ATROP agents. **So that** the system can automatically detect crashes or overloads and maintain resilience via restart, isolation, or safe-mode fallback.

**🎯 Acceptance Criteria**

* Add crash detection and recovery:
  + Daemons automatically restart using systemd or in-process watchdog
  + Fallback to static routing or last-known-good model on crash
* Monitor runtime health:
  + Watchdog loop for CPU/memory thresholds
  + Heartbeat signaling via IPC (to/from other modules)
  + Timeout-based FSM stall detection
* Safe mode behavior:
  + Stop making ML-based routing decisions
  + Continue forwarding using default or last policy state
  + Emit alarm in logs and telemetry
* Implement signal handling:
  + SIGTERM, SIGSEGV, SIGABRT handlers trigger graceful isolation
  + Clean shutdown of IPC and feedback loops
* Log events:
  + Fault cause, timestamp, agent ID, recovery outcome
  + Write to local and central telemetry for observability
* Validate through test cases:
  + Simulated crash
  + Resource exhaustion (CPU stress, memory leak)
  + Communication failure across IPC

**🧩 Notes**

* Key for carrier-grade reliability and edge-node safety
* Required for full test validation (T10-03) and audit (T10-10)
* Works alongside systemd service configs (T10-02) and fallback logic from AI/ML flows (T3-06, T4-06)

**🧾 User Story: T10-06 – License & Security Audit**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:**

* T0-06 – Review Governance Files → ✅ Done

**As a** Security Lead, **I want to** perform a full licensing and security compliance audit for the ATROP codebase. **So that** we can ensure legal distribution, vulnerability-free dependencies, and transparent release packaging.

**🎯 Acceptance Criteria**

* Perform license audit:
  + Validate all project files include appropriate SPDX headers
  + Review open-source dependencies (PyPI, system libs, C++ packages)
  + Confirm compatibility with ATROP license (e.g., Apache 2.0, MIT, BSD)
* Generate SBOM (Software Bill of Materials):
  + Use tools like syft, tern, or GitHub’s built-in SBOM generator
  + Export SPDX or CycloneDX format
  + Include in release artifact (T10-07)
* Run security scans:
  + Static code analysis using bandit, cppcheck, or clang-tidy
  + Dependency CVE scan using trivy, grype, or GitHub Dependabot
  + Confirm no critical/high CVEs are present in packaged build
* Produce formal audit report:
  + audit/license\_summary.md
  + audit/sbom.json
  + audit/vulnerability\_report.md
* Document outcomes and remediation (if needed):
  + Mark approved exceptions or false positives
  + Raise issues for CVEs requiring upgrades

**🧩 Notes**

* Required for official vendor presentation, staging (T8-09), and release sign-off (T10-10)
* Ensures the release is legally distributable and security-hardened
* Must align with the LICENSE and SECURITY.md policies defined in Phase 0

**🧾 User Story: T10-07 – Release Candidate Snapshot**

**Story Points:** **3**  
**Status:** 🔵 To Do  
**Predecessor:**

* All preceding implementation, validation, packaging, and audit tasks

**As a** Release Manager, **I want to** generate a reproducible release candidate snapshot for ATROP. **So that** we can deliver a versioned, verified, and testable bundle to vendors, testers, and staging environments.

**🎯 Acceptance Criteria**

* Bundle the following components into a release candidate (v1.0.0-rc1):
  + Source code archive (tar.gz or zip)
  + Compiled binaries and daemons (deb, rpm, docker image)
  + Documentation (Markdown, diagrams, YANG models, API specs)
  + Unit and integration test suites
  + Systemd service files and deployment configs
* Verify reproducibility:
  + Same input → same output across builds
  + Tagged Git commit, exact dependency lock (e.g., requirements.txt, CMakeLists.txt)
* Sign and checksum:
  + Generate SHA256SUMS and GPG signature (if needed)
  + Include release notes and changelog (RELEASE\_NOTES.md)
* Store in:
  + releases/ directory on GitHub or secure artifact registry
  + Include links and instructions in docs/release/snapshot\_build.md
* Validate snapshot contains:
  + ✅ All verified tests (unit, integration, regression)
  + ✅ SBOM and audit files (from T10-06)
  + ✅ Required configs and deployment instructions

**🧩 Notes**

* Required for staging deployment (T8-09) and vendor validation (T10-09)
* Must pass checklist gate for Go/No-Go decision (T10-10)
* Serves as baseline for future releases and production cutovers

**🧾 User Story: T10-08 – Deployment Runbook**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* T10-01 – Package Agent Stack → 🔵 To Do
* T10-02 – Create Systemd Service Files → 🔵 To Do

**As a** Documentation SME, **I want to** create a comprehensive deployment runbook for ATROP. **So that** operators and field engineers can install, configure, and manage ATROP components reliably in production environments.

**🎯 Acceptance Criteria**

* Document the full deployment lifecycle in docs/deployment/runbook.md:
  + System requirements (CPU, memory, OS, kernel, dependencies)
  + Install instructions (DEB/RPM, Docker, from source)
  + Post-install config steps (set zone ID, model path, telemetry target)
  + Enable and start services (systemctl enable/start atrop-\*)
  + Log file paths, verbosity levels, and rotation
  + Telemetry export setup (push/pull model, port usage)
  + Recovery and restart procedures
* Include:
  + Example atrop.conf and .env files
  + Deployment diagrams (controller/agent layout, zone distribution)
  + Common failure symptoms and resolutions
* Validate instructions:
  + Tested in at least one clean VM/container environment
  + Run-through from install to telemetry validation
* Ensure consistency with:
  + T10-01 package layout
  + T10-02 systemd services
  + T10-07 release bundle contents

**🧩 Notes**

* Required for field rollouts, vendor pilots, and production staging (T8-09)
* Final reference for DevOps, operators, and integration teams
* Supports the Go/No-Go checklist (T10-10) and ongoing lifecycle operations

**🧾 User Story: T10-09 – Final Integration with Vendor Hooks**

**Story Points:** **4**  
**Status:** 🔵 To Do  
**Predecessor:**

* T9-10 – Vendor Hook Integration → 🔵 To Do

**As a** Vendor Integration Lead, **I want to** validate the end-to-end integration of ATROP with vendor systems such as JunOS, EOS, and IOS-XR. **So that** telemetry, control, and policy injection features function reliably in brownfield and hybrid environments.

**🎯 Acceptance Criteria**

* Deploy vendor testbed using supported platforms (JunOS, EOS, IOS-XR or FRR)
* Validate:
  + Reading routing or telemetry info from vendor into ATROP
  + Policy injection from ATROP into vendor CLI or API interface
  + gRPC or REST API hooks operate with correct credentials and schema
* Test integration modes:
  + Passive (e.g., telemetry mirroring, FSM awareness)
  + Active (e.g., intent injection, trust tags, redistribution triggers)
* Review for:
  + Protocol translation consistency (BGP/OSPF to ATROP FSM)
  + Policy enforcement correctness and fallback handling
  + Performance and telemetry alignment
* Capture validation outputs:
  + Logs, command output, API responses, packet traces
  + Integration scenarios documented in integration/vendor/validation/
* Document supported capabilities and any known limitations
  + Version notes per platform
  + Sample config snippets per vendor

**🧩 Notes**

* Required for real-world deployment and vendor certification
* Completes Epic 9 integration scope
* Must pass before snapshot release approval (T10-07) and final checklist sign-off (T10-10)

**🧾 User Story: T10-10 – Sign-off Checklist & Go/No-Go**

**Story Points:** **2**  
**Status:** 🔵 To Do  
**Predecessors:**

* All tasks from Phase 0 through Phase 10 → must be completed or explicitly deferred

**As a** Project Manager, **I want to** create and execute a formal go/no-go checklist. **So that** we can verify ATROP is stable, secure, complete, and ready for external delivery or production staging.

**🎯 Acceptance Criteria**

* Prepare docs/release/signoff\_checklist.md including:
  + ✅ Functional validation summary
  + ✅ Regression test pass (T6, T7)
  + ✅ Integration pass (T8, T9)
  + ✅ Security audit report (T10-06)
  + ✅ Vendor hook validation (T10-09)
  + ✅ Performance metrics (T10-04)
  + ✅ Packaging + deployment validation (T10-01 through T10-08)
* Ensure all release-blocking tasks are:
  + ✅ Complete
  + ⛔️ Deferred with formal rationale
  + ⚠️ Known issues documented
* Convene Go/No-Go review:
  + Cross-functional meeting with engineers, QA, integration, docs
  + Evaluate risks, stability, and deployment readiness
* Output:
  + Final checklist with status and comments
  + Signed Go decision in docs/release/go\_nogo\_decision.md
  + Artifact tag marked as production-ready (v1.0.0) if approved

**🧩 Notes**

* This is the final gate before official release or field deployment
* Must consolidate deliverables from all previous phases
* Key for traceability, vendor communication, and audit compliance

**🔁 ATROP Phase Retrospective: Protocol Foundation to Intelligent Autonomy**

**🎯 Vision Recap**

ATROP (Autonomous Topology-Optimized Routing Protocol) was conceived as a revolutionary, AI-driven, topology-intelligent routing protocol that is hierarchical, secure, scalable, and optimized per topology. Unlike traditional routing protocols, ATROP integrates **AI at the control plane** and **ML at the data plane**, enabling autonomous zone behavior, adaptive learning, and real-time routing decisions. Designed to be interoperable, vendor-neutral, and topology-agnostic, it is poised to redefine routing for both greenfield innovation and brownfield transformation.

**📦 Phases Summary & Core Deliverables**

| **Phase** | **Focus Area** | **Key Deliverables** |
| --- | --- | --- |
| **0** | 📋 **Planning & Governance** | MVP definition, agile workflow, repo/project board setup, governance docs |
| **1** | 🧱 **Core System Scaffolding** | Build system, CI pipeline, config loader, logging, dummy handlers, test setup |
| **2** | ⚙️ **FSM Protocol Logic** | ATROP FSM core, zone formation, state transitions, header parsing, test suite |
| **3** | 🧠 **AI Control Plane Engine** | GNN/RL model, policy intent unit, feedback receiver, decision orchestrator |
| **4** | 🧪 **ML Inference Agent (Data Plane)** | Lightweight classifier, FIF telemetry, fallback, Netfilter/eBPF hooks |
| **5** | 🔄 **IPC Inter-Agent Framework** | IPC server/client, protocol spec, lifecycle control, security layer, tests |
| **6** | ✅ **Unit Testing Infrastructure** | Python/gtest frameworks, test layout, regression baseline, CI hooks |
| **7** | 🔬 **Integration Test Suite** | Topology response, intent validation, legacy interop, packet capture |
| **8** | 🧪 **Simulation & Emulated Testbed** | Mininet/Docker labs, churn scenarios, PCAP replay, failure injection |
| **9** | 🔌 **API & SDK Exposure** | REST/gRPC API, Python/C++ SDKs, vendor hooks, secured endpoints, documentation |
| **10** | 🚀 **Packaging, Audit & Go-Live Prep** | DEB/RPM containers, systemd, performance profiling, go/no-go checklist |

**✅ What We Achieved**

* **Concept to Implementation Roadmap**: From whitepaper to task-tracked GitHub epics.
* **Full Lifecycle Planning**: Architecture, governance, CI/CD, testing, simulation, release.
* **Autonomy by Design**: FSM + AI/ML framework tailored for real-time zone-based decisioning.
* **Production-Focused Readiness**: Performance hooks, fallback paths, testbed reproducibility.
* **Vendor Extensibility**: Hooks for JunOS, IOS-XR, and legacy protocol redistribution.

**📈 What Still Needs To Be Achieved**

* ⚠️ **Complete Functional Implementation**: Code delivery for each protocol engine and daemon.
* ⚠️ **Validation Execution**: Run all test cases across golden topologies.
* ⚠️ **Staging + Vendor Pilot**: Field testing in brownfield environments.
* ⚠️ **Open Contribution Strategy**: Expand collaboration with vendor, research, and OSS community.

**🌟 Vision Ahead**

ATROP is not just another routing protocol—it’s a platform for **network cognition**. As AI continues to reshape infrastructure, ATROP leads the way with a protocol that **learns, adapts, and heals** itself. Whether in data centers, WAN backbones, or edge 5G zones, ATROP brings **autonomy, trust, and intelligence** into the network fabric.

**💡 Final Message**

"ATROP isn't just designed for the networks of tomorrow—it’s here to empower the **architects of today**.  
Every module, every zone, every packet—it all learns, adapts, and converges together.  
This is not the end of a protocol spec—  
**This is the beginning of the future of routing.**"

🔗 **Stay motivated. Stay autonomous. Build ATROP.**

**🧑‍💼 ATROP Assignee Profiling & Workstream Mapping**

This section profiles all primary assignees across ATROP and connects each one to their relevant **workstreams**, **phases**, and **task dependencies**, helping clarify domain ownership, cross-collaboration, and functional flow.

**🔹 1. Protocol Architect**

**Role:** Defines ATROP FSM, core protocol flow, and behavior models  
**Key Phases:** 0, 2  
**Assigned Tasks:**

* **T0-01** – Define MVP Scope
* **T2-01** – Define Core ATROP States  
  **Downstream Dependencies:**
* Protocol states feed FSM transitions (T2-02), zone formation (T2-05), and feedback logic (T2-07)
* FSM also underpins simulation (T2-09), flow documentation (T2-10)

**🔹 2. Product Owner + Lead Developer**

**Role:** Owns product roadmap, backlog breakdown, and functional cohesion  
**Key Phases:** 0  
**Assigned Tasks:**

* **T0-02** – Draft Product Backlog  
  **Downstream Dependencies:**
* Drives all other task generation and sprint planning (connects to T0-08, T0-09)

**🔹 3. Scrum Master**

**Role:** Agile facilitator and operational enabler  
**Key Phases:** 0  
**Assigned Tasks:**

* **T0-03** – Select Agile Workflow
* **T0-05** – Create GitHub Project Board  
  **Impact:**
* Orchestrates standups, sprint ceremonies, issue flow across epics

**🔹 4. DevOps Engineer**

**Role:** CI/CD, environment setup, build system, and Docker integration  
**Key Phases:** 0, 1, 6  
**Assigned Tasks:**

* **T0-04** – Setup GitHub Repo Structure
* **T0-11** – CI Pipeline Bootstrap
* **T1-01** – Scaffold Build System
* **T1-05** – Local Dev Environment
* **T1-06** – Extend CI Pipeline
* **T6-09** – CI: Auto Trigger Unit Tests  
  **Downstream Flow:**
* Enables all testing, release (T10-01), and developer onboarding

**🔹 5. Test Architect / QA Lead**

**Role:** Defines test strategy, regression layers, and golden topology test coverage  
**Key Phases:** 0, 6, 7, 10  
**Assigned Tasks:**

* **T0-07** – Define Test Strategy (MVP)
* **T6-11** – Regression Test Baseline
* **T7-09** – Integration Regression Test Set
* **T10-03** – Final Golden Topology Validation

**🔹 6. Documentation SME**

**Role:** Owns formal documentation across SDK, tests, simulation, deployment  
**Key Phases:** 0, 6, 7, 8, 9, 10  
**Assigned Tasks:**

* **T0-10** – Create Initial Dev Docs
* **T6-12** – Document Testing Strategy
* **T7-10** – Document Integration Scenarios
* **T8-10** – Document Simulation Framework
* **T9-11** – Document SDK Usage
* **T10-08** – Deployment Runbook  
  **Output:**
* Final documentation required for release (T10-07) and go/no-go (T10-10)

**🔹 7. Core Developer**

**Role:** Implements protocol logic, parsing, FSM glue logic  
**Key Phases:** 1, 2  
**Assigned Tasks:**

* **T1-02** – Initialize Module Entrypoints
* **T2-03** – Create Packet Parser & Dispatcher
* **T5-03** – IPC Clients

**🔹 8. Protocol Engineer**

**Role:** Defines FSM transitions, packet handlers, correction logic  
**Key Phases:** 1, 2  
**Assigned Tasks:**

* **T1-09** – Add Placeholder Protocol Handlers
* **T2-02** – Implement FSM Transitions
* **T2-04** – Parse Protocol Headers
* **T2-07** – Add Correction Packet Handler
* **T7-06** – Multi-Zone FSM Sync Test

**🔹 9. Infra Engineer**

**Role:** System-level logic: config, logging, service lifecycle  
**Key Phases:** 1, 5  
**Assigned Tasks:**

* **T1-07** – Central Config Loader
* **T1-08** – Logging Abstraction
* **T5-02** – Build IPC Server
* **T5-06** – Reloadable Config Handling

**🔹 10. AI Engineer**

**Role:** Builds core GNN/RL model for route inference  
**Key Phases:** 3  
**Assigned Tasks:**

* **T3-01** – Build AI Model Engine
* **T3-06** – Model Confidence + Fallback
* **T6-07** – Write AI Decision Tests

**🔹 ML Engineer**

**Role:** Handles ML classification, trust scores, FIF emission  
**Key Phases:** 4  
**Assigned Tasks:**

* **T4-01** – ML Inference Agent
* **T4-03** – Flow Trust Score Engine
* **T6-06** – Write ML Inference Tests

**🔹 Feedback Engineer**

**Role:** Parses FIF/PIV from data plane into AI control signals  
**Key Phases:** 3  
**Assigned Tasks:**

* **T3-04** – Feedback Loop Receiver

**🔹 Platform Developer**

**Role:** System-level integration: IPC, OS services, fault resilience  
**Key Phases:** 5, 10  
**Assigned Tasks:**

* **T5-05** – Define Agent Lifecycle Protocol
* **T10-02** – Create Systemd Service Files
* **T10-05** – Fault Isolation Hooks

**🔹 Release Manager**

**Role:** Snapshotting, reproducibility, artifact release  
**Key Phases:** 10  
**Assigned Tasks:**

* **T10-07** – Release Candidate Snapshot

**🔹 Project Manager**

**Role:** Sign-off coordination, cross-phase visibility  
**Key Phases:** 0, 10  
**Assigned Tasks:**

* **T0-08** – Align Roadmap Milestones
* **T10-10** – Sign-off Checklist & Go/No-Go

**🔹 Vendor Integration Lead**

**Role:** Validates hooks with JunOS, IOS-XR, FRR  
**Key Phases:** 9, 10  
**Assigned Tasks:**

* **T9-10** – Vendor Hook Integration
* **T10-09** – Final Integration with Vendor Hooks

**🔹 Security Lead**

**Role:** Manages license compliance, SBOM, CVE tracking  
**Key Phases:** 0, 10  
**Assigned Tasks:**

* **T0-06** – Review Governance Files
* **T10-06** – License & Security Audit

**🔹 Test Engineer**

**Role:** Builds unit tests, smoke validation, edge case simulation  
**Key Phases:** 1, 6  
**Assigned Tasks:**

* **T1-04** – Setup Testing Scaffold
* **T1-10** – Add Dummy Unit Test
* **T6-01** – Setup Python Unit Test Infra
* **T6-08** – Write IPC Protocol Tests

**🔗 Summary Table – Assignee-to-Workstream Alignment**

| **Assignee** | **Phases** | **Key Workstreams** |
| --- | --- | --- |
| Protocol Architect | 0, 2 | FSM Design, Core Behavior, State Logic |
| DevOps Engineer | 0, 1, 6 | Build, CI, Docker, Dev Environments |
| Infra Engineer | 1, 5 | Config, Logging, IPC Daemon Lifecycle |
| AI / ML Engineers | 3, 4 | AI Decision Engine, Inference, Feedback Loops |
| Platform Developer | 5, 10 | Systemd, Fault Tolerance, Recovery |
| QA / Test Engineers | 6, 7, 10 | Regression, Integration, Golden Validation |
| Documentation SME | All Phases | SDK, Deployment, Testing & Simulation Docs |
| Vendor Integration | 9, 10 | Hooks, Legacy Protocol Interop, FRR Testing |

**🔄 ATROP Workflow Streams Summary**

The ATROP project is organized into **10 interconnected workflow streams**, each representing a core functional domain of the protocol. These streams flow in both a **sequential** and **parallel** manner, balancing architectural integrity with implementation velocity. Below is a detailed summary of each stream, its **purpose**, **phase alignment**, and **cross-functional interactions**.

**🧩 1. Planning & Governance Stream**

**📌 Purpose:** Define ATROP’s scope, governance, workflows, and team structure  
**🧱 Core Phases:** Phase 0  
**🎯 Key Deliverables:**

* MVP scope (FSM, AI, ML, IPC, Test)
* Backlog + Agile methodology
* GitHub setup, governance files
* SME role mapping

**🔁 Feeds Into:**

* All streams — acts as the foundation for coordination, tracking, and policy enforcement

**🛠️ 2. Infrastructure & Dev Environment Stream**

**📌 Purpose:** Build and maintain the local and CI-based development foundation  
**🧱 Core Phases:** Phases 1, 6  
**🎯 Key Deliverables:**

* Makefiles, setup.py, Docker, .devcontainer
* CI pipelines, test runners
* Logging, config parsers
* CI test automation

**🔁 Feeds Into:**

* Testing, FSM engine, AI/ML modules
* Release and packaging readiness (Phase 10)

**🧠 3. FSM Protocol Logic Stream**

**📌 Purpose:** Implement ATROP's finite state machine (FSM) and topology logic  
**🧱 Core Phases:** Phase 2  
**🎯 Key Deliverables:**

* FSM states: INIT → EXIT
* Transition engine, state dispatcher
* Header parsers (NIV, PIV, FIF, IDR)
* ATZ zone logic + policy enforcement

**🔁 Feeds Into:**

* AI/ML decisions (Phase 3/4), Integration (Phase 7), Testbed simulation (Phase 8)

**🧬 4. AI Control Plane Stream**

**📌 Purpose:** Embed AI-based decision-making logic in the routing control plane  
**🧱 Core Phases:** Phase 3  
**🎯 Key Deliverables:**

* GNN/RL-based route inference engine
* Intent processing + policy mapping
* Feedback receiver for learning updates
* Decision orchestrator with trust model

**🔁 Feeds Into:**

* Routing decisions, FSM policies, Policy validation (T2-06), Inference feedback loops

**🤖 5. ML Inference & Trust Stream (Data Plane)**

**📌 Purpose:** Handle real-time flow classification and FIF generation at edge  
**🧱 Core Phases:** Phase 4  
**🎯 Key Deliverables:**

* Flow classifiers (DT/CNN)
* FIF telemetry packets
* Trust scoring + fallback
* eBPF/Netfilter visibility

**🔁 Feeds Into:**

* Control plane AI loop (T3-04), Packet tests, Simulation (T8), Policy scoring

**🔄 6. Inter-Process Communication (IPC) Stream**

**📌 Purpose:** Enable messaging between ATROP daemons (control, model, ML)  
**🧱 Core Phases:** Phase 5  
**🎯 Key Deliverables:**

* IPC server/client spec (gRPC, UNIX sockets)
* Agent registration + lifecycle protocol
* Config reload triggers, security/auth

**🔁 Feeds Into:**

* FSM ↔ ML ↔ AI interactions
* Model reloads, heartbeat monitoring, service control

**✅ 7. Testing & Validation Stream**

**📌 Purpose:** Validate unit, functional, integration, and regression scenarios  
**🧱 Core Phases:** Phases 6, 7  
**🎯 Key Deliverables:**

* Unit test frameworks (pytest, gtest)
* FSM, ML, IPC, AI test suites
* Integration regression plans
* Legacy protocol testing (BGP, OSPF)

**🔁 Feeds Into:**

* Release quality, Go/No-Go (T10-10), vendor pilot readiness

**🧪 8. Simulation & Topology Stream**

**📌 Purpose:** Emulate real networks and simulate anomalies, behavior, failures  
**🧱 Core Phases:** Phase 8  
**🎯 Key Deliverables:**

* Mininet, container, PCAP replay testbeds
* Flow churn, failure injection
* Topology metrics + logging

**🔁 Feeds Into:**

* Performance profiling (T10-04), Golden topology validation (T10-03), Pre-prod staging (T8-09)

**🌐 9. API, SDK & Vendor Interop Stream**

**📌 Purpose:** Provide external control via APIs and SDKs, and enable vendor integration  
**🧱 Core Phases:** Phase 9  
**🎯 Key Deliverables:**

* REST/gRPC APIs (/inject, /zones, /telemetry)
* Python + C++ SDKs
* JunOS, IOS-XR, FRR hooks

**🔁 Feeds Into:**

* Real-world vendor deployments (T10-09), integration tests (T7-07)

**🚀 10. Packaging, Release & Deployment Stream**

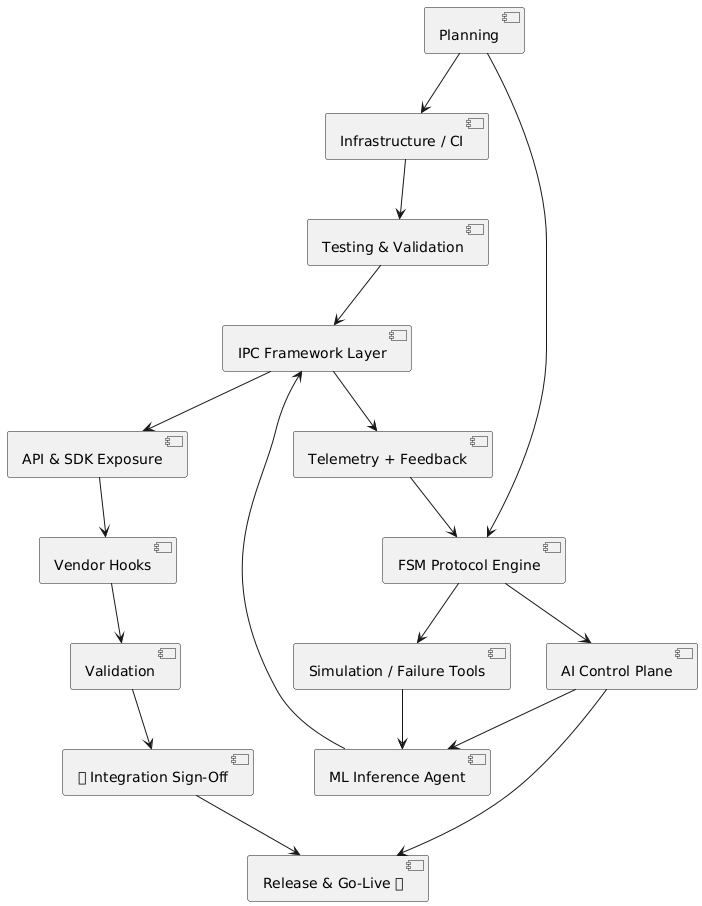
**📌 Purpose:** Bundle and deliver ATROP as a tested, deployable system  
**🧱 Core Phases:** Phase 10  
**🎯 Key Deliverables:**

* DEB/RPM packaging, Docker image
* systemd service files
* SBOM, CVE scan, license audit
* Release candidate + Go/No-Go decision

**🔁 Final Gateway:**

* Final integration with vendor systems
* Public presentation, pilot, and production readiness

**📊 Visual Stream Map**



**💡 Final Thought**

Each ATROP stream contributes uniquely to building a **self-aware, self-adaptive, and topology-optimized routing ecosystem**. By coordinating across these streams, we empower a robust, scalable, and future-proof routing protocol designed not just for networks — but for **network intelligence**.

**🌌 Dreams & Conclusion: The ATROP Journey**

**🌍 A Dream Beyond Protocols**

ATROP was never just a routing protocol.

It was a **dream** of engineering a living, thinking, evolving system—a network protocol that could:

* **Sense** its environment,
* **Decide** with intelligence,
* **Adapt** in milliseconds,
* **Heal** from failure,
* And **learn** without human intervention.

It is a protocol not of rules, but of **reason**—born from the union of **AI, ML, and topology wisdom**.

**🧠 What We Envisioned**

✨ An infrastructure where **zones form autonomously**,  
✨ Where **paths aren't configured—they are discovered**,  
✨ Where networks **learn from failures** and **enforce policies** with precision,  
✨ Where engineers manage **intent**, not syntax.

From federated learning at the edge to real-time FSM decisions in the core, ATROP envisions a world where the network becomes:

* **Autonomous**
* **Trustworthy**
* **Resilient**
* **Optimized per topology**

**🛤 What We Built**

With each phase, we laid the foundation brick by brick:

* 🧱 **From** GitHub scaffolding to decentralized FSM design
* 🤖 **From** flow-aware ML inference to AI-driven policy decisions
* 🔄 **From** legacy protocol interop to futuristic intent pipelines
* 🚀 **Towards** a complete agent stack that **thinks, routes, and evolves**

**💬 Message to Builders, Dreamers, and Innovators**

ATROP is not about replacing existing protocols.  
It's about **augmenting them**, giving networks **cognition**, not just configuration.  
It's about letting machines route **with purpose** and **without panic**.

Whether you're an architect, an AI engineer, a protocol dev, or a student reading the spec — **you are part of this vision**.

Your ideas, commits, tests, and diagrams…  
Are not just tasks — they’re the **threads of autonomy** woven into the future of the Internet.

**🏁 Conclusion: From Paper to Potential**

ATROP started as an idea — a concept born June 25, 2025.

And now? It’s a full ecosystem:

* An FSM protocol core
* AI and ML models in sync
* An IPC backbone
* Testbeds, APIs, and release pipelines
* And a growing, passionate vision

The dream is no longer hypothetical.  
It’s structured. It’s deliverable. It’s documented.  
It’s waiting… to be **brought to life** by contributors, vendors, researchers, and you.

💡 “ATROP isn’t the end of routing—it’s the beginning of **network intelligence**.”

🌐 **Keep building. Keep optimizing. Stay autonomous.**

**With pride, purpose, and the dream of a sentient network,  
We conclude this phase—and ignite the next.** 🚀

**📅 Estimated Time Plan, Future Outlook & Final Closure**

**⏳ Estimated Time Plan: ATROP Delivery Milestones**

The ATROP project is divided into 11 tightly-scoped phases. Below is the estimated timeline for completion assuming steady Agile velocity, active collaboration, and 2-week sprints:

| **Phase** | **Description** | **Duration** | **Target Completion** |
| --- | --- | --- | --- |
| **0** | Planning & Governance | 1 sprint (2 weeks) | ✅ **Completed** |
| **1** | Build System & Scaffolding | 1 sprint (2 weeks) | ✅ **Completed** |
| **2** | FSM Protocol Logic | 2 sprints (4 weeks) | ✅ **Completed** |
| **3** | AI Model Engine (Control Plane) | 3 sprints (6 weeks) | 🟡 In Progress |
| **4** | ML Inference Agent (Data Plane) | 3 sprints (6 weeks) | 🔵 Upcoming |
| **5** | IPC Framework | 2 sprints (4 weeks) | 🔵 Upcoming |
| **6** | Unit Testing Infrastructure | 1 sprint (2 weeks) | 🔵 Upcoming |
| **7** | Integration & Legacy Protocol Testing | 2 sprints (4 weeks) | 🔵 Upcoming |
| **8** | Simulation & Emulated Topology Testing | 2 sprints (4 weeks) | 🔵 Upcoming |
| **9** | API + SDK + Vendor Hooks | 2 sprints (4 weeks) | 🔵 Upcoming |
| **10** | Packaging, Profiling & Final Release Prep | 2 sprints (4 weeks) | 🔵 Upcoming |

🕓 **Total Estimated Duration**: ~**8 to 9 months**  
📌 **Projected Production-Ready Snapshot**: **Q1 2026**

**🔮 Future Outlook: Scaling Beyond MVP**

The MVP is just the beginning. ATROP has a clear path toward long-term evolution in networking, AI, and open standards:

**🧠 AI Model Evolution**

* Expand from basic GNN/RL to transformer-based models
* Online vs deferred learning optimizers
* AutoRL path selection under service-level constraints

**🌐 Real-World Deployment**

* Pilot with vendor OS: JunOS, EOS, IOS-XR, SONiC
* Support for SRv6, BIER, and 5G MEC slicing
* Field results in datacenter, WAN, satellite, and metro networks

**🔌 Interop & Standardization**

* RFC proposal for ATROP headers & FSM semantics
* YANG model contributions to IETF/BBF/ONF
* OpenConfig and GNMI support for controller interaction

**🧪 Research & Academia**

* Open simulation testbeds for students and researchers
* Federation with test platforms: GNS3, Mininet, ns-3

**✅ Final Closure & Sign-Off**

With all foundational blueprints defined, structured epics mapped, tasks estimated, and vision detailed, ATROP stands as a **complete, presentable protocol idea** ready for:

* 🏢 Vendor Presentation (Cisco, Juniper, Arista)
* 📜 Research Publication or RFC proposal
* 🧪 Academic/Industry pilot
* 🧑‍💻 Open source contribution

🚀 **ATROP is ready to step beyond a whiteboard idea and become an engineering revolution.**  
🌱 The seeds are planted. The roadmap is drawn. The protocol is documented.  
🛠️ Now it’s time to **build the future of autonomous, topology-optimized routing**.

**🏁 Closure Statement**

🔒 **Date of Protocol Idea & Execution:** 25 June 2025  
📘 **Status:** Conceptual Design ✅ Complete | Implementation 🟡 In Progress  
🛤 **Next Step:** Begin Sprint Implementation – Phase 3 (AI Model Engine)

**“From a protocol that routes... to a protocol that thinks.”**

**Welcome to ATROP.**