End‑to‑End Infrastructure for Story‑Based Testing with Behavioral Programming (BP)  
— Comprehensive Build & Operations Guide —

Version 1.0 • Audience: Engineers, QA Leads, SREs, Product Analysts

# Abstract

This document specifies a complete, reproducible method to build a story‑based testing infrastructure using Behavioral Programming (BP) from an OpenAPI contract (JSON or YAML). The system converts API contracts into testable narratives (“stories”), generates deterministic baselines, leverages an LLM for non‑deterministic evaluation, and outputs human‑readable artifacts and a life‑cycle summary. The guide is self‑contained and assumes no prior knowledge of BP or LLM pipelines.

# 1. Conceptual Background

## 1.1 Behavioral Programming (BP)

Behavioral Programming decomposes desired system behavior into concurrently executed, scenario‑driven threads (“behavior threads” or b‑threads). Each thread expresses a story (e.g., ‘user signs up and receives a confirmation e‑mail’). At runtime, the system selects events that satisfy threads’ requests while avoiding conflicts. In testing, BP allows us to model stories as executable constraints that interleave with the system under test (SUT).

## 1.2 Story‑Based Testing

Story‑based testing models expected interactions as readable narratives that map to concrete API calls and assertions. Stories provide coverage over realistic, user‑facing flows rather than isolated endpoints.

## 1.3 Deterministic vs. Non‑Deterministic Evaluation

* Deterministic (DET): Rules and gold answers are fixed; grading is repeatable.
* Non‑Deterministic (NONDET): An LLM interprets outcomes using rubrics; grading tolerates ambiguity and contextual variation.

# 2. Scope & Objectives

1. Accept OpenAPI contracts (JSON/YAML) as the canonical input (SUT & Real‑World systems).
2. Generate DET baselines (gold, prompts, graders).
3. Configure a model key and optionally train an evaluation model for NONDET grading.
4. Run E2E pipelines that produce artifacts and a lifecycle summary (lifecycle.js).
5. Provide governance: versioning, reproducibility, security, and CI‑friendly operation.

# 3. Prerequisites & Tooling

|  |  |  |
| --- | --- | --- |
| Domain | Requirement | Notes |
| OS | Windows 10/11 (cmd.exe) | Batch orchestration uses .bat files. |
| Python | 3.10+ | Use system Python or virtualenv. |
| Git | Latest | For versioning and GitHub integration. |
| Model Access | OPENAI\_API\_KEY (or equivalent) | Export in shell; never commit secrets. |
| Network | HTTPS access to raw files | To fetch Real‑World OpenAPI via manifest. |

# 4. Repository Architecture

## 4.1 Directory Layout (Representative)

scripts\  
 llm\  
 run\_llm\_gold\_and\_grade.py # core gold/grade runner  
 realworld\  
 manifest.json # list of RW systems & spec URLs  
 fetch\_openapi\_manifest.bat # fetches RW OpenAPI files  
 training\  
 e2e\_train\_7suts\_real\*.bat # orchestrates SUT + RW DET/NONDET flows  
 pipelines\  
 run\_det\_all\_7suts.bat # DET for 7 SUTs  
packs\  
 7\_suts\<name>\openapi.json # in-repo SUT specs  
 realworld\<system>\openapi.(json|yaml|yml) # fetched RW specs  
artifacts\  
 det\_checked\ # outputs of deterministic runs  
 nondet\_checked\ # outputs of model-based runs  
README.md  
.gitignore

## 4.2 Data & Artifact Contracts

|  |  |  |  |
| --- | --- | --- | --- |
| Artifact | Producer | Purpose | Format / Location |
| OpenAPI Spec | Fetcher / Provided | Canonical API contract | JSON or YAML @ packs/<category>/<name>/openapi.\* |
| Gold & Prompts | DET Generator | Deterministic baseline | JSON/TSV under artifacts/det\_checked/... |
| Grading Reports (DET) | Runner | Fixed‑rule evaluation | JSON/CSV under artifacts/det\_checked/... |
| Model Config | Operator | Key, model IDs | Env vars; config in .bat files |
| NONDET Reports | E2E + Model | LLM‑assessed outcomes | JSON/CSV under artifacts/nondet\_checked/... |
| Lifecycle Summary | Aggregator | Readable flow visualization | lifecycle.js + HTML assets |

# 5. End‑to‑End Build Process (Detailed)

## 5.1 Input Capture

1. Collect OpenAPI contracts for all targets.
2. For built‑in SUTs, verify presence under packs/7\_suts/\*/openapi.json.
3. For Real‑World (RW) systems, ensure entries exist in scripts/realworld/manifest.json (name, raw URL, format).

Deliverables: OpenAPI files in JSON or YAML. Stored under packs/7\_suts/\* (SUT) and packs/realworld/<system> (RW).

## 5.2 Real‑World Fetch (Manifest‑Driven)

Run the fetcher to materialize RW specs:

scripts\realworld\fetch\_openapi\_manifest.bat

* Reads scripts/realworld/manifest.json.
* Downloads OpenAPI files to packs/realworld/<system>/openapi.(json|yaml|yml).
* Skips or logs 404s; validate URLs are raw JSON/YAML, not HTML.

Deliverables: RW OpenAPI files; normalized folder structure for downstream steps.

## 5.3 Normalization & Validation

* Optional: validate OpenAPI (e.g., spectral / openapi-cli).
* Ensure security schemes, servers, and paths are parseable.
* Identify non‑CRUD endpoints that require special stories.

Deliverables: a validation report (optional); updated contracts if issues were fixed.

## 5.4 Deterministic (DET) Dataset Generation

1. For SUTs: run scripts/pipelines/run\_det\_all\_7suts.bat.
2. For RW: run scripts/training/e2e\_train\_7suts\_real\*.bat to trigger DET for selected systems.
3. The runner (scripts/llm/run\_llm\_gold\_and\_grade.py) extracts endpoints, creates gold prompts and grading templates.

Deliverables: artifacts/det\_checked/<category>/<name>/\* — gold, prompts, fixed‑rule grading outputs in JSON/CSV/TSV.

## 5.5 Model Access & Keys

* Set provider key in shell (e.g., set OPENAI\_API\_KEY=sk‑...).
* Choose evaluation model (e.g., gpt‑4o‑mini) in batch config.
* Do NOT commit secrets; add .env and keys to .gitignore.

Deliverables: valid environment configuration for subsequent model‑based runs.

## 5.6 Model Training (Optional)

* If supported, run training to produce a model ID for NONDET grading.
* Record the model ID in your config (.bat or env vars).

Deliverables: trained model ID (string), training logs, and stored hyperparameters if applicable.

## 5.7 End‑to‑End Execution & NONDET Grading

1. Execute E2E scripts to run stories against the SUT/RW systems.
2. Collect outputs under artifacts/nondet\_checked/<category>/<name>/\*
3. LLM rubric checks produce flexible pass/fail with rationale.

Deliverables: NONDET JSON/CSV reports; model inferences; discrepancies versus DET for investigation.

## 5.8 Readable Interface & Lifecycle

* Aggregate execution into a lifecycle summary (lifecycle.js).
* Render a readable interface (HTML+JS) from artifacts to support triage and reviews.

Deliverables: lifecycle.js, optional static HTML; screenshots for reports, and shareable artifacts.

# 6. Operating Procedures

## 6.1 Daily Operation (Runbook)

|  |  |  |
| --- | --- | --- |
| Task | Command | Expected Outputs |
| Fetch RW contracts | scripts\realworld\fetch\_openapi\_manifest.bat | packs/realworld/<system>/openapi.\* |
| DET for 7 SUTs | scripts\pipelines\run\_det\_all\_7suts.bat | artifacts/det\_checked/7suts\_\* |
| DET for RW | scripts\training\e2e\_train\_7suts\_real\*.bat | artifacts/det\_checked/realworld\_\* |
| E2E + NONDET | scripts\training\e2e\_train\_7suts\_real\*.bat | artifacts/nondet\_checked/\* |

## 6.2 File Naming & Formats (Authoritative)

|  |  |  |
| --- | --- | --- |
| File / Folder | Format | Notes |
| openapi.json | JSON | Canonical SUT contract |
| openapi.yaml / openapi.yml | YAML | Canonical RW contract |
| \*.gold.json | JSON | DET gold story specifications |
| \*.prompts.json | JSON | Prompt packs for grading |
| \*.grade.json/csv | JSON/CSV | DET or NONDET grading results |
| lifecycle.js | JavaScript | Top‑level lifecycle aggregation |

## 6.3 Quality Gates

* All OpenAPI documents must be schema‑valid (no critical lint errors).
* DET must run without fatal errors for SUTs before E2E.
* NONDET must record model ID and version in metadata.

# 7. Governance: Security, Reproducibility, Versioning

* Secrets never committed; .gitignore includes .env, keys, artifacts.
* Artifacts built from exact OpenAPI versions; pin RW URLs with commit SHAs when possible.
* Include run metadata (timestamp, model ID, commit hash) in outputs for forensic analysis.

# 8. CI/CD Integration (Optional)

1. Trigger fetch + DET on PRs touching OpenAPI files.
2. Upload artifact summaries as CI artifacts.
3. Gate merges on passing DET; mark NONDET as informational unless critical.

# 9. Troubleshooting

|  |  |  |
| --- | --- | --- |
| Symptom | Cause | Resolution |
| '... was unexpected at this time' in .bat | Unescaped | or ( ) in blocks | Escape | as ^|; avoid parentheses in echo inside FOR/IF blocks |
| HTTP 404 during fetch | URL points to HTML or moved file | Use raw file URLs; verify with curl |
| Python not found | PATH / venv mismatch | where python; ensure %VIRTUAL\_ENV%\Scripts\python.exe exists or fall back |
| Model key missing | Env var not set | set OPENAI\_API\_KEY=... before running |

# 10. Appendices

## A. Sample .gitignore

/.venv/  
\_\_pycache\_\_/  
\*.pyc  
\*.pyo  
\*.pyd  
node\_modules/  
dist/  
build/  
.DS\_Store  
Thumbs.db  
\*.swp  
\*.swo  
.env  
.env.\*  
\*.key  
\*.pem  
\*.crt  
secrets\*/  
artifacts/  
logs/  
/tmp/  
.old/  
\*/.old/

## B. GitHub Publishing (Step‑By‑Step)

1. Create an empty GitHub repository (no README/license).
2. From the project root: git init; git add .; git commit -m "Initial commit"
3. git branch -M main; git remote add origin https://github.com/<you>/<repo>.git
4. git push -u origin main

## C. Manifest Entry Template

{  
 "systems": [  
 { "name": "stripe", "url": "https://.../openapi.yaml", "format": "yaml" }  
 ]  
}