LEX TRI – Temporal Agent

LEX TRI is a tri‑temporal debugging and remediation agent designed to run alongside the EXO AGI overlay. It maps anomalies across three synchronized timelines—Valid Time (VT), Transaction Time (TT) and Decision Time (DT)—to clarify whether issues stem from stale data, ingestion lag, or premature decisions. When integrated into your swarm, LEX TRI can diagnose defects, propose guarded fixes, and archive all traces in a temporal knowledge graph.

## Core Features

* **Tri‑Temporal Debugging** – builds VT/TT/DT maps for anomalies and visualizes divergence across the three axes.
* **Runtime Hooks** – attaches to Python, Rust and FastAPI services to collect traces, variables and logs.
* **Guarded Fix Generation** – produces reversible patches complete with patch.diff, updated tests, rollback instructions and reasoning stored in TRACE.json.
* **CI/CD Integration** – blocks merges on regression, auto‑opens issues and creates fix PRs when authorized via the LEXTRI\_WRITE\_OK secret.
* **Audit Trails** – archives all traces and resolutions in a temporal knowledge graph for auditability.
* **Multi‑Environment Support** – runs as a GitHub Action, Visual Studio Code/JetBrains plugin, Copilot Studio conversational agent and self‑hosted container.

## Security

LEX TRI operates in two modes:

1. **Diagnostics (default)** – read‑only mode that builds tri‑temporal traces and attaches diagnostics to pull requests. No writes are performed.
2. **Guarded Write** – enabled only when the repository secret LEXTRI\_WRITE\_OK is set to true. In this mode LEX TRI can apply patches and open fix PRs, but never writes directly to the main branch. All changes go through a guarded PR flow requiring human approval and passing checks.

## Installation

To add LEX TRI to your repository:

1. Copy the workflow file lextri.yml into .github/workflows/.
2. Copy the action.yml, Dockerfile, requirements.txt, lextri\_runner.py and lextri\_config.yml into a directory of your choice (for example lextri/).
3. Ensure that your repository contains the LEXTRI\_WRITE\_OK secret if you want to enable guarded fixes. Without this secret the workflow will run in diagnostics‑only mode.
4. Commit these files and push them to your repository. On pull requests, the diagnostics job will execute and attach reports to the PR. If write mode is enabled and LEX TRI proposes a fix, it will open a guarded PR for review.

## Usage

LEX TRI is designed to be queryable via GitHub PR comments, IDE integrations and Copilot Studio. Some example interactions include:

* **"Show why balances diverged between TT and DT yesterday."** – LEX TRI generates a report explaining the divergence.
* **"Generate a guarded fix for the routing off‑by‑one error."** – LEX TRI proposes a patch with tests and rollback instructions.
* **"List anomalies in the risk module since March 1 with VT/TT/DT."** – LEX TRI returns a list of anomalies with tri‑temporal context.
* **"Replay yesterday’s trades for misalignments."** – LEX TRI runs a temporal replay to find misalignments.
* **"Prepare a fix but keep it suggest‑only."** – LEX TRI generates a patch without applying it.

## JetBrains Integration

LEX TRI runs natively in JetBrains IDEs via the GitHub Actions integration and Qodana code quality scanning. You can view tri‑temporal diagnostics in the IDE’s Problems tool window and click through to the offending code. The provided qodana\_code\_quality.yml workflow can run alongside lextri.yml to perform static analysis using JetBrains Qodana. Together, they offer rich temporal debugging and conventional code quality feedback within IntelliJ IDEA.