Proposal Summary

Base repository URL: eth-llm-poc/blob/main

This document is the submission-ready proposal summary for the Ethereum Foundation RFP on integrating LLMs into protocol security research. It is grounded in a working system delivered in the eth-llm-poc repository, with the installable Python package `eip-verify`. It describes a low-risk, auditable path to full execution and consensus coverage in 4–6 months.

Executive Summary

Ethereum's Protocol Security team manually audits multiple client implementations against evolving specifications—a labor-intensive process that scales poorly with fork velocity and client diversity. We deliver **eth-llm-poc**, a working LLM-assisted verification system that automates obligation extraction, spec mapping, and client gap detection, producing auditable discrepancy reports.

What exists today: - A multi-phase pipeline covering execution-specs and Geth, running in CI with structured artifacts. - Validated runs across EIP-1559, EIP-2930, and EIP-7702 with Claude Opus 4.5. - Reusable GitHub Actions workflows enabling per-EIP or batch execution.

What the proposal delivers: - Phase 1–2: Pipeline hardening, quantitative accuracy baselines, execution client matrix. - Phase 3–4: Consensus-specs ingestion, consensus client coverage, EL/CL linkage. - Phase 5–6 (optional): CI gating, quality dashboards, broader protocol security mapping.

Architecture rationale: We evaluated multi-agent systems, RAG pipelines, and symbolic repo maps. All introduced instability, unpredictable results, and poor auditability. The chosen direct-chained architecture prioritizes reproducibility and clear audit trails—critical for security infrastructure.

Key Differentiators

- **Working system today:** eth-llm-poc runs end-to-end with CI integration and auditable artifacts—not a proposal for future work.
- **Auditable by design:** strict phase boundaries, versioned runs, and structured outputs (CSV, JSON, Markdown) enable independent verification.
- **Proven accuracy:** Claude Opus 4.5 produces plausible outputs with low observed false positives in qualitative validation; quantitative baselines are Phase 1 work.
- **Clear scaling unit:** `eip-verify` CLI enables single-EIP or batch execution; the same unit extends to EIP × client matrices.

Evidence and Validation

Evidence is documented via qualitative evaluation (manual spot-checks and cross-model comparison) and multiple pipeline runs. Formal quantitative accuracy baselines are scoped for Phase 1.

Current validation approach: - Manual spot-checks of obligations against EIP text, execution-specs, and Geth sources. - Cross-model comparison (Haiku, Sonnet, Opus) to assess stability and quality differences. - Model selection based on observed quality: Opus 4.5 produces the most plausible outputs; Haiku showed high noise in client mapping.

Evidence sources: - Qualitative evaluation summary: docs/QUALITATIVE_EVALUATION.md - Validation transcript: examples/qualitative_validation_transcript.md - Example CI runs (EIP-7702, Opus): - Run 21571909617 - Run 21570420032 - Local runs: EIP-1559 (Sonnet, Haiku, Opus), EIP-2930 (Opus) in `examples/runs/`

> **Note on determinism:** Phase inputs, outputs, and artifacts are deterministic in structure. LLM responses remain stochastic. We achieve reproducibility via version pinning, cross-model validation, and fake mode for regression testing.

---

1. RFP Objectives and How We Meet Them

These objectives are validated by the working system. The table below maps each RFP requirement to concrete outputs.

RFP Objective	How We Address It	Evidence
Automated spec compliance	Extract obligations, map to spec + client code, produce gap reports	Run artifacts + indexed CSVs
Workflow integration	Reusable GitHub Actions workflow with auditable artifacts	CI workflows + job manifests
Efficiency and accuracy	Phased pipeline with model selection based on observed quality	Phase manifests + evaluation notes

---

2. Why This Approach

Ethereum's spec and client surface is broad and evolves frequently. The right solution is the most **reproducible and auditable**, not the most complex.

Approaches evaluated: | Approach | Finding | | — | — | | Multi-agent systems | Coordination complexity, harder reproducibility, opaque failures | | RAG pipelines | Context drift, retrieval failures, fragile context windows | | Symbolic repo maps | High engineering overhead, unpredictable results | | LangChain deep agent / aider repomap | More layers without accuracy gains |

Chosen approach: Direct chained agent calls with strict phase boundaries using Claude Agent SDK. Minimal layers between LLM and codebase preserves traceability. This is the lowest-risk path to a trustworthy workflow.

---

3. Scope Boundaries

In scope (Phase 1–4): - Execution-specs ingestion and EIP obligation extraction. - Execution client validation (starting with Geth; client matrix planned). - Consensus-specs ingestion and consensus client validation. - Deterministic artifacts, manifests, and summary reports.

Out of scope (unless explicitly requested): - Automatic code changes or patches. - Formal verification tooling beyond structured discrepancy reports. - Production deployment inside client release pipelines.

---

4. System Overview (Ingest → Analyze → Report)

Pipeline flow:

1. **Ingest** → EIP markdown, execution-specs, client repos
2. **Phase 0A** → Extract obligations from EIP
3. **Phase 1A/1B** → Locate and analyze spec implementations
4. **Phase 2A/2B** → Locate and analyze client implementations
5. **Report** → CSV indices, manifests, summary reports

Output	Purpose
`obligations_index.csv`	Spec-side obligation mapping
`client_obligations_index.csv`	Client-side mapping and gaps
`run_manifest.json`	Per-phase metadata for audit
`summary.md` / `summary.json`	Human + machine readable reports

Design principles: simplicity, auditability, reproducibility.

---

5. Technical Approach (Methodologies, Frameworks, Tools)

Methodology: Direct chained agent calls with strict phase boundaries and deterministic artifacts.

Frameworks and models: | Category | Decision | Rationale | | — | — | — | | Primary agent | Claude Agent SDK | Best performance and reliability observed | | Primary model | Claude Opus 4.5 | Highest accuracy in validation | | Fallback model | Claude Sonnet 4.5 | Cost-effective for batch runs | | Other models tested | GPT-5.2, Gemini 3 Pro | Lower quality for this task | | Agent frameworks tested | LangChain Deep Agent, Aider + RepoMap | More complexity, weaker results |

Tools: Native filesystem and CLI tools with structured outputs and manifest metadata.

---

6. Deliverables and Acceptance Criteria

Each deliverable has a clear acceptance criterion so EF can validate progress without ambiguity.

Deliverable	Acceptance Criteria	Evidence
Technical architecture & design	Architecture covers ingest, analysis, report, and toolchain	Architecture doc + system diagrams
Working prototype	eth-llm-poc runs per-EIP pipeline on Geth with artifacts	CLI pipeline + run outputs
Integration guidelines	Reusable workflow integration documented	Workflow usage + examples
Operations & extension	Setup, maintenance, and future phases documented	Ops guide + extension plan

---

7. Success Metrics (Initial Targets)

Targets are refined with EF in Phase 1. Current targets reflect feasible outcomes for a 4–6 month roadmap.

Metric	Initial Target	Current State	Measurement
Coverage	100% of selected EIPs per fork mapped	EIP-1559, EIP-2930, EIP-7702 validated	CSV indices + manifests
Accuracy	<=5% false positives after Phase 1 tuning	Qualitative: Opus shows low observed FP	Ground truth dataset + precision/recall in Phase 1
Reproducibility	100% artifact completeness per run	Achieved in current runs	Manifests + structured outputs
Throughput	200 runs/month baseline	CI batch runs functional	Batch workflow logs
Run time	<=60 minutes per CI run	~30 min observed for EIP-7702	CI run logs

> **Accuracy note:** Current validation is qualitative (spot-checks, cross-model comparison). Phase 1 establishes quantitative baselines with curated ground truth for 2–3 well-understood EIPs.

---

8. Project Plan and Timeline (4–6 Months)

The plan expands coverage in a controlled way: first harden the pipeline and establish accuracy baselines, then scale across execution and consensus layers.

Phase	Timing	Dependencies	Outputs
Phase 0 (complete)	Done	—	Working CLI, reusable workflow, run artifacts
Phase 1	Month 1	None	Ground truth dataset, accuracy baselines, prompt tuning
Phase 2	Month 2	Phase 1 accuracy >=80%	Execution client matrix (3+ clients), batch coverage
Phase 3	Month 3	Parallel to Phase 2	Consensus-specs ingestion, obligation extraction

Phase	Timing	Dependencies	Outputs
Phase 4	Month 4	Phases 2 and 3	Consensus client matrix, EL/CL linkage
Phase 5 (optional)	Month 5	—	CI gating, quality thresholds, dashboarding
Phase 6 (optional)	Month 6	—	Extended phases for broader protocol security mapping

Critical path: Phase 1 accuracy validation gates Phase 2 expansion.
Contingency: Phases 5–6 absorb schedule slip from core phases if needed.

Detailed plan: see Supporting Materials (Project plan and timeline).

---

9. Evaluation Criteria (RFP)

Criterion	Evidence	Future Expansion
Scalability	Single EIP or batch across a fork; `eip-verify` scales in CI	Client matrices, parallel runs
Accuracy	Opus 4.5 yields plausible outputs; quantitative baselines in Phase 1	Prompt tuning, evaluators, expanded validation
Reliability	Simple chained pipeline with deterministic outputs	Harnesses, logs, monitoring
Security	Runs inside CI; report-only outputs; minimal surface	Tighter sandboxing as needed

---

10. Risks and Mitigations

We intentionally surface real failure modes observed during development.

Risk	Observed Evidence	Mitigation
LLM hallucination (confident but incorrect mappings)	Haiku runs showed high noise in client locations (many ABI/test file false positives)	Model selection (Opus primary), cross-model review, ground truth regression
Spec ambiguity (multiple valid interpretations)	Some obligations (e.g., EIP1559-OBL-030) appear questionable relative to spec text	Citation-based extraction, manual arbitration workflow, disputed cases flagged
Extraction incompleteness (missing obligations)	Opus runs occasionally missed constraint details	Cross-EIP comparison, manual review of edge cases, iterative prompt tuning
Model drift (provider updates change behavior)	Not yet observed	Pinned model versions, regression suite, phased rollout of updates
Cost overruns	Token usage varies by EIP complexity	Budget monitoring, `--max-turns` limits, Sonnet fallback for batch runs
Client variation (patterns LLM doesn't recognize)	Different file structures across clients	Per-client prompt tuning, constrain to core paths, false negative tracking

---

11. Budget and Cost Structure (EUR)

We provide a cost model separating engineering effort from operational costs. Opus is used for high-fidelity runs; Sonnet for cost-effective batch coverage.

Cost Item	Estimate
Solo delivery (4 months, discounted)	EUR 53,760
Solo delivery (6 months, discounted)	EUR 80,640
LLM runs (Opus, 200/month)	EUR 6,751/month
LLM runs (Sonnet, 200/month)	EUR 4,051/month
CI (Linux baseline, 200 runs)	EUR 81/month

Cost controls: - `--llm-mode fake` for zero-cost CI and regression runs. - `--max-turns` to limit conversation length. - Phase selection to avoid unnecessary model calls.

Full cost model: see Supporting Materials (Budget and cost structure).
CSV breakdown: see Supporting Materials (Budget CSV).

---

12. Vendor Background

Petros Lambropoulos is an independent consultant with 13 years of experience in software engineering, ML systems, and production-grade AI.

Career highlights: - Workable (2016–2019): Senior Software Engineer, NLP team — resume parsing and job matching systems. - **NannyML (2021–2023):** Senior Software Engineer — ML monitoring platform for model drift detection. - **Recent consulting:** Hedera/CNO (compliance-first tokenization infrastructure), dikaio.ai (agentic workflows and evaluation pipelines).

This project: Delivered eth-llm-poc end-to-end with installable `eip-verify` CLI, CI workflows, and validated runs.

Vendor links:
- Website: https://petroslamb.github.io/peterlamb/
- Blog: https://lambpetros.substack.com/
- GitHub profile: https://github.com/petroslamb
- GitHub repo: https://github.com/petroslamb/eth-llm-poc
- LinkedIn: https://uk.linkedin.com/in/petroslamb

Docs: see Supporting Materials (Vendor background) and resume.

---

13. Assumptions and Dependencies

- Access to forked execution and consensus client repositories in the project.
- Ability to run GitHub Actions workflows for batch jobs.
- EF feedback cadence on scope selection and acceptance criteria.

---

14. Supporting Materials (Repository Links)

Append the folder and file below to the base repository URL above.

- Technical architecture
  Folder: docs/proposal/
  File: TECHNICAL_ARCHITECTURE_AND_DESIGN.md
- Project plan and timeline
  Folder: docs/proposal/
  File: PROJECT_PLAN_AND_TIMELINE.md
- Budget and cost structure
  Folder: docs/proposal/
  File: BUDGET_AND_COST_STRUCTURE.md

- Budget CSV
  Folder: docs/proposal/
  File: BUDGET_AND_COST_STRUCTURE.csv
- Integration guide
  Folder: docs/proposal/
  File: INTEGRATION_GUIDE.md
- Operations and extension
  Folder: docs/proposal/
  File: OPERATIONS_AND_EXTENSION.md
- Evaluation criteria response
  Folder: docs/proposal/
  File: EVALUATION_CRITERIA_RESPONSE.md
- Vendor background
  Folder: docs/proposal/
  File: VENDOR_BACKGROUND_AND_REFERENCES.md
- Proposal readiness checklist
  Folder: docs/proposal/
  File: PROPOSAL_READINESS_CHECKLIST.md
- Canonical RFP
  Folder: docs/proposal/
  File: Request for Proposal (RFP)_ Integrating Large Language Models (LLMs) into Ethereum Protocol Security Research.md