GEDCOM Parser Project – Next Phase Execution Plan

This document captures the **immediate next steps and execution plan** for advancing the GEDCOM_Parser_Project from its current operational v1.0 foundation toward the **v1.1 milestone**. It is derived from a structural analysis of the repository, existing outputs, logs, tests, and the consolidated roadmap and audit materials.

---

1. Current State Assessment (What You Have Now)

1.1 Architecture Status

The project is already **architecturally sound and modular**: - Loader → Tokenizer → Tree Builder → Entity Builders → Post-processing → Export - Strong separation of concerns under `src/gedcom_parser/` - Configuration-driven normalization via `config/` and `datasets/` - Rich post-processing pipeline (entity resolution, scoring, XREF resolution) - Comprehensive test suite validating most parser components

Key conclusion: You are *past* the experimental phase. This is now a **hardening, completeness, and compliance phase**, not a rewrite.

1.2 Pipeline Maturity

Layer	Status
File loading & encoding	✔️Functional
Tokenization & segmentation	✔️Stable
Tree construction	✔️Stable
Entity extraction	✔️Strong
Name normalization	✔️Advanced
Event parsing	⚠ Partial
Source / citation handling	⚠ Partial
Validation enforcement	⚠ Weak
Persistence model	⚠ Incomplete
Performance & scaling	⚠ Untuned

---

2. Strategic Shift for the Next Phase

Primary Goal (v1.1)

> **Elevate the parser from "works well" to "fully GEDCOM 5.5.5 compliant, auditable, and scalable."**

This phase is about: - Completeness - Correctness - Repeatability - Long-run maintainability

---

3. Immediate Next Steps (Do These First)

STEP 1 — Freeze Behavior & Baseline (Week 0)

Objective: Protect what already works.

Actions: 1. Run full test suite (`verify_all.sh`) and capture results 2. Run parser against: - `mock_files/gedcom.ged` - `mock_files/utf16.ged` 3. Archive outputs as **v1.0 baseline artifacts** 4. Tag repo state (`git tag v1.0-baseline`)

Why: - Prevents regressions during refactoring - Gives you a comparison point for v1.1 correctness

---

STEP 2 — Formalize the Processing Contract (Critical)

Objective: Define exactly what the parser *must* produce.

Deliverables: - `schemas/output_schema.json` - `schemas/entity_schema.json`

Define: - Required vs optional fields - Allowed nulls / placeholders - Cardinality (single vs list) - Referential integrity rules

Why this matters: - Becomes the **source of truth** for validation, persistence, and exports - Enables automated schema validation pre/post parse

---

STEP 3 — GEDCOM Completeness Audit (Parser Gap Closure)

Objective: Ensure *all* GEDCOM 5.5.5 constructs are parsed.

High-priority gaps to close: - Multiple NAME blocks per INDI - Full name piece extraction: - NPFX, SPFX, GIVN, NICK, NSFX - Nickname parsing (quotes / parentheses) - EVEN records (custom & standard events) - SOUR, REPO, NOTE, OBJE coverage

Concrete action: - Add explicit coverage tests per GEDCOM tag - Fail fast on unknown/invalid tags (log + continue)

---

4. Data Model & Persistence (Next Critical Layer)

STEP 4 — Database Strategy Reset

Objective: Make storage trustworthy and per-file isolated.

Actions: 1. Implement **one database per GEDCOM file** 2. Enforce foreign keys (INDI ↔ FAM ↔ SOUR) 3. Restore omitted schema tables: - places - citations - associations - media

Deliverable: - `schema_validator.py` - `db_factory.py`

---

5. Normalization & Enrichment Refactor

STEP 5 — Extract Enrichment into First-Class Module

Actions: - Create `src/gedcom_parser/enrichments/` - Move: - phonetics - romanization - name variants - Drive behavior from `enrich_names_config.yml`

Add: - Toggleable enrichment stages - Batch processing for performance - Diacritic preservation guarantees

---

6. Validation & Observability

STEP 6 — Validation as a Pipeline Gate

Implement validation at: - Pre-parse (file structure & encoding) - Post-parse (schema compliance) - Pre-export (referential integrity)

Add: - `parsing_errors.json` - Record-level error context (line number, xref)

---

7. Performance & Scale Preparation

STEP 7 — Make Large Files Safe

Actions: - Streaming file reads (where possible) - Batch inserts & exports - Optional multiprocessing for enrichment - Progress reporting every N records - Log rotation per run

---

8. Refactoring Discipline

STEP 8 — Controlled Cleanup

Rules: - No refactor without a test - One responsibility per module - Config over code - Remove legacy scripts after coverage exists

Tools to add: - flake8 - mypy - profiling hooks

---

9. Execution Order (Recommended)

1. Baseline freeze
2. Output schema contract
3. GEDCOM tag completeness
4. Database schema & validation
5. Enrichment refactor
6. Validation gates
7. Performance tuning
8. Documentation & CLI polish

---

10. Final Guidance

You are no longer "figuring it out".

This project is now in **engineering mode**, not exploration. The architecture is solid — the next phase is about: - Precision - Standards compliance - Operational confidence

If you want, the next step can be: - A **task-by-task coding plan** - A **single-module deep dive** (e.g., name parsing or entity resolution) - A **v1.1 checklist turned into GitHub issues**

This plan intentionally prioritizes correctness over novelty. That is exactly what a genealogical parser must do.