

Claude Code Bootstrap – robolawyer-tm Ecosystem

> **Purpose**: This document is the single entry point for Claude Code to understand the robot

1. System Goal

Develop semantics that work with conflict data – to predict beneficial outcomes from construct

Core Concept: Analogical Synthesis (not Dialectical)

The system operates on a **left/right duality**:

- **Left (Semantic)**: Analogical, *logos*-based – language, emotion, empathy, felt meaning. T
- **Right (Digital)**: Analytical – code, filesystem organization, pattern detection, objectiv

Critical distinction: The desired output is analogical – rooted in human emotion, empathy-

2. Existing Infrastructure

2.1 Repositories

Repo	Location	Status	Purpose
pillars	~/repos/pillars`	Active	Architecture vision, rules, design docs (this rep
star-bridge	GitHub: robolawyer-tm	Active	SSH admin connection manager, phone disco
secret-server	On Android phone / GitHub	Active MVP	Hardened Flask app, autovivified

2.2 Current Stack

Component	Technology
Core Language	Python 3
Web Framework	Flask (on phone)
Encryption	SSH/OpenSSH, PBKDF2-HMAC-SHA256
Filesystem Mounting	SSHFS
Android Environment	Termux + Termux:Boot
Data Storage	JSON over filesystem (autovivified)
AI Assistance	Claude, Perplexity, Gemini

2.3 Phone Architecture (Star Topology)

...

Linux Laptop (admin)

-
- SSH tunnel → Phone:8022 (sshd)
- SSHFS mount → ~/secret-server/android_mnt/
- Port forward → localhost:5001 (Flask)

Phone (Termux):

- ~/secret-server/
- web_server.py # Flask app (port 5001, localhost only)
- main.py
- lib/
- ■■■ auth.py
- ■■■ crypto.py

```

■ ■■■ network.py
■■■ db/
■ ■■■ sys_adm/auth.json
■ ■■■ {username}/{app_name}/secret.json
■■■ start_server.sh
...

```

2.4 What Already Works

- ■ Hardened Android userspace (Termux, auto-start, wake-lock)
- ■ SSH tunnel + SSHFS mounting
- ■ Secret manager with `deep\_update` autovivification
- ■ Human-in-the-loop pairing (hotspot + w3m confirmation)
- ■ Filesystem-as-database pattern (JSON, no opaque formats)

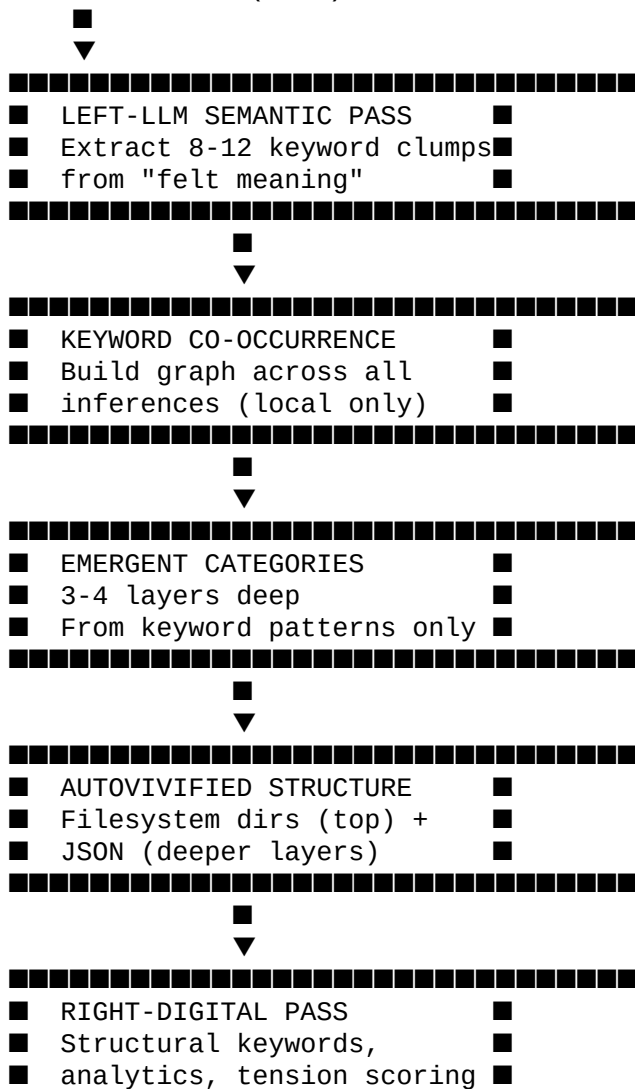
3. What Needs Building: The Vivify Inference Pipeline

This is the primary development target. Everything below must be built from scratch.

3.1 Pipeline Overview

...

Raw Inferences (text)



4.1 Inference Unit

Each inference is atomic:

```
```json
{
 "id": "inf_XXX",
 "timestamp": "ISO-8601",
 "source": "origin identifier",
 "raw_text": "the full inference text",
 "left_keywords": [],
 "right_keywords": [],
 "clumps": {},
 "category_paths": [],
 "guardrail_actions": {}
}
```
```

****Rule 1.1**:** The only material used to generate categories is the inference text itself plus

4.2 Left-LLM Semantic Pass

****Rule 2.1**:** Read the full inference. Generate 8-12 ****keyword clumps**** that "feel" central to

****Rule 2.2**:** Keywords must be concept-level tokens, not surface words:

- ■ `conflict\_asymmetry`, `perjury\_pattern`, `therapeutic\_potential`, `emotional\_truth`
- ■ `lie`, `unfair`, `thing`, `stuff`

****Rule 2.3**:** Never force keywords into any predefined ontology. All grouping is emergent.

****Rule 2.4**:** Normalize: lowercase, `\_` for spaces, strip punctuation.

****Rule 2.5**:** Merge obvious near-duplicates locally, but do not assume global synonymy.

4.3 Co-occurrence Graph

****Rule 3.1**:** Nodes = keywords. Edges = co-occurrence in the same inference. Weight = count of

****Rule 3.2**:** Minimum co-occurrence threshold of 2+ inferences for category eligibility.

****Rule 3.3**:** Graph is local-only – built purely from collected inferences.

4.4 Emergent Categories (3-4 layers)

****Rule 4.1**:** Category seeds are keywords with high degree and/or high edge weights.

****Rule 4.2**:** Sub-categories form from strongly connected neighborhoods around seeds.

****Rule 4.3**:** Maximum depth: 3-4 layers.

****Rule 4.4**:** Category names drawn directly from existing keywords – no invented labels.

****Rule 4.5**:** Keywords may appear in multiple category paths (graph, not tree). No forced sing

4.5 Assignment

****Rule 5.1**:** An inference belongs to a category path if it contains the seed keyword + at lea

****Rule 5.2**:** Multi-assignment allowed. No single home enforced.

****Rule 5.3**:** Unmatched inferences go to `unclustered/` for future passes.

4.6 Iterative Refinement

****Rule 6.1**:** New inferences can create new seeds, split or merge old categories.

****Rule 6.2**:** No external schema is ever imported. All changes result from updated keyword pat

****Rule 6.3**:** Guardrails constrain *how* categories form but never introduce foreign taxonomie

4.7 Self-Evolution

****Rule 7.1**:** Live workers continually refine:

- The rules themselves (as `.md` files)
- Categorization and summarization in the vivified database
- Configuration (as `.json` files)

****Rule 7.2**:** Rules are agentically developed – the system proposes rule updates based on obser

5. Guardrails (Humanity Rules)

Guardrails are preprocessing constraints that keep autovivification aligned with the project's

```
```json
{
 "guardrails": {
 "no_external_taxonomies": {
 "description": "Block any import of foreign ontologies or predefined category trees",
 "action": "reject"
 },
 "logos_analog_priority": {
 "description": "Prioritize analogical/emotional keyword clumps over purely analytical or",
 "priority_clumps": ["emotional_truth", "analogic_logic", "therapeutic_potential"],
 "action": "boost"
 },
 "strict_duality": {
 "description": "Maintain left/right separation in processing; synthesis only at the merge",
 "action": "enforce_split"
 },
 "no_digital_reduction_of_analog": {
 "description": "Never reduce felt meaning to purely computational terms",
 "action": "block"
 }
 }
}
```
```

6. Dual-View Storage

Each inference maintains separate left and right keyword lists:

```
```json
{
```

```

 "id": "inf_123",
 "raw_text": "...",
 "left_keywords": ["conflict_asymmetry", "emotional_truth", "therapeutic_potential"],
 "right_keywords": ["json_indexing", "pattern_detection", "similarity_clustering"],
 "clumps": {
 "conflict_resolution": ["conflict_asymmetry", "resolution_focus"],
 "therapeutic_signal": ["emotional_truth", "therapeutic_potential"]
 },
 "category_paths": ["conflict_resolution/therapeutic_signal"],
 "tension_score": 0.87
}

```

**\*\*Tension score\*\***: Measures divergence between left and right views. High tension = where digi

---

### ## 7. Filesystem-as-Database Principles

- The filesystem **is** the database. No opaque binary formats.
- Top 2-3 layers = directories (human-navigable)
- Deeper layers = JSON (machine-traversable, autovivified)
- Users can audit data with any text editor
- Backups are `cp`. Migrations are `mv`.
- Git tracks structural evolution

---

### ## 8. Design Philosophy: Structure → Process → Validation

All development follows this strict order:

1. **\*\*Define the TARGET FILE STRUCTURE first\*\*** (non-negotiable)
2. **\*\*Write PROCESS documentation\*\*** that explicitly references that structure
3. **\*\*Include VALIDATION CODE\*\*** so correctness can be checked at any point

See: `DESIGN\_PHILOSOPHY.md` for full rationale.

---

### ## 9. Development Priorities (Build Order)

#### ### Phase 1: Core Engine

1. `lib/vivify\_core.py` – Autovivification engine (Perl-style hash-of-hashes in Python)
2. `lib/inference.py` – Inference data model + I/O
3. `lib/keyword\_graph.py` – Co-occurrence graph structure
4. `lib/index.py` – Master index builder

#### ### Phase 2: Pipeline Scripts

5. `scripts/extract\_keywords.py` – Left-LLM keyword extraction (needs LLM API)
6. `scripts/build\_cooccurrence.py` – Graph builder from keyword lists
7. `scripts/categorize.py` – Emergent category detection from graph
8. `scripts/autovivify.py` – Create filesystem + JSON structures

#### ### Phase 3: Analysis

9. `scripts/right\_pass.py` – Digital/analytical keyword extraction
10. `scripts/tension\_score.py` – Left/right divergence measurement
11. `scripts/search.py` – Query across vivified structures

### ### Phase 4: Integration

12. Integration with secret-server (Flask endpoints for vivified data)
13. Guardrail enforcement engine
14. Self-evolution mechanism (rule update proposals)

---

## ## 10. Reference Documents

All in `~/repos/pillars/`:

File	Contains
`PILLARS_SUMMARY.md`	Full architectural vision
`PILLARS_SUMMARY.html`	Published web version
`DESIGN_PHILOSOPHY.md`	Structure → Process → Validation methodology
`TECHNICAL_AUDIT_SUMMARY.md`	Current state of secret-server + Vivify pillars
`Semantic_edge_manifesto.md`	Secret-server README / feature list
`Inferences_Categorizing and summarizing inference.md`	Detailed Perplexity conversation
`Pillars - Semantic intent.md`	System goal + synthesis-without-antithesis rationale
`PILLARS_SUMMARY_UPDATE.md`	Latest Mission section refinements
`vivify-payload.tree.txt`	Payload creation/retrieval flow
`MESH_TOPOLOGY_VISION.md`	Star → mesh networking vision
`NOTATION_VISION.md`	Process flow notation design
`AnalogicalSynthesis-IntuitiveOutcomes.md`	Deep dive on analogical synthesis

---

## ## 11. Constraints and Non-Negotiables

1. **Local-first**: Zero cloud dependencies. Everything runs on participant hardware.
2. **Privacy-preserving**: All data stays local. SSH tunnels for transport.
3. **No external taxonomies**: All structure emerges from the data.
4. **Analogical orientation**: Output serves human emotion and empathy, not objective reduction.
5. **Filesystem transparency**: No opaque formats. Human-readable JSON.
6. **Python 3**: Core language for all new development.
7. **Structure first**: Define target file layout before writing any process code.

---

## ## 12. Getting Started (for Claude Code)

```bash

- # 1. Read this document fully
- # 2. Read PILLARS_SUMMARY.md for ecosystem context
- # 3. Read the Inferences document for detailed rule development history
- # 4. Begin with Phase 1: build vivify_core.py and inference.py
- # 5. Follow Structure → Process → Validation at every step

```

**First task**: Create the `vivify/` directory structure, implement `lib/vivify\_core.py` with

---

\*© robolawyer-tm • Local-first • Privacy-preserving • Human-centric\*