

Base120: A Six-Family Taxonomy of 120 Reasoning Operators for Structured Human and Agent Decision-Making

Authors: Reuben Bowlby (ORCID: 0009-0002-5620-1103), HUMMBL, LLC

Repository: <https://github.com/hummbbl-io/base120>

Version: 2.0.0 (SDK) / 1.0.0 (canonical registry)

License: Apache 2.0

Abstract

Mental models — the cognitive frameworks humans use to interpret problems and make decisions — have been discussed extensively in the management and cognitive science literature, but have never been formalized as an executable software library with a structured taxonomy. Existing resources fall into three categories: content sites that explain models but offer no programmatic access; curated link lists with no executable tooling; and platform-specific skill packs that cannot be used outside their host environment. We present Base120, a canonical registry of 120 reasoning operators organized into six transformation families (Perspective, Inversion, Composition, Decomposition, Recursion, Systems), with a stdlib-only Python SDK, a CLI, an MCP (Model Context Protocol) server, and an append-only JSONL ledger for governance-readable decision records. The six-family taxonomy provides a navigation map for selecting which mode of reasoning to apply, a completeness check for ensuring a mode has been exhausted, and a composition grammar for chaining operators across families in structured decision-making arcs. We describe the taxonomy’s design rationale, the canonical registry format, the SDK API, and three use cases: structured human decision-making, AI agent reasoning via MCP, and multi-agent coordination with persisted audit trails. Base120 is deployed in production as the cognitive layer of the HUMMBL multi-agent architecture.

1. Introduction

The concept of mental models — internal representations of external reality that humans use to reason, predict, and decide — has a long history in cognitive science (Johnson-Laird [1]) and management practice (Charlie Munger’s “latticework of mental models” [2]). The practical literature recommends that decision-makers build a diverse collection of mental models and learn when to apply each one. However, this literature has three gaps that Base120 addresses:

1. **No structured taxonomy.** Existing mental-models resources present flat lists of 50–100 models. A flat list provides no navigation map for selecting which model to apply, no completeness check for ensuring a mode of thinking has been exhausted, and no composition grammar for chaining models in sequence. Base120 organizes its 120 operators into six transformation families based on the type of cognitive transformation each operator performs, providing all three.
2. **No executable library.** Content sites (e.g., Farnam Street [3]) explain mental models but offer no programmatic access. Curated link lists (e.g., awesome-concepts [4]) provide no executable tooling. Platform-specific skill packs (e.g., cc-thinking-skills [5]) cannot be used outside their host environment. Base120 is a general-purpose Python library with zero runtime dependencies, a CLI, an MCP server, and a cross-language YAML registry that can be read by any language.
3. **No audit trail.** Existing resources treat mental models as one-time cognitive acts with no persistent record. Base120’s append-only ledger persists every operator application as a JSONL tuple, enabling retrospective review of which reasoning was applied, when, and with what confidence — a prerequisite for governance-readable decision records in multi-agent systems.

Base120 is deployed in production as the cognitive layer of the HUMMBL multi-agent architecture [6], where it provides the reasoning operator vocabulary for agent decision-making and the ledger format for governance audit trails.

Contributions

1. We present a six-family taxonomy of 120 reasoning operators, with design rationale for the family boundaries and operator selection (Section 3).
2. We describe the canonical registry format — a single YAML file that is the source of truth for all 120 operators, readable by any language (Section 4).
3. We describe the stdlib-only Python SDK, CLI, and MCP server that provide multi-surface access to the registry (Section 5).
4. We describe the append-only ledger format for governance-readable decision records (Section 6).
5. We present three use cases: structured human decision-making, AI agent reasoning via MCP, and multi-agent coordination with audit trails (Section 7).

2. Background

2.1 Mental Models in Cognitive Science

Johnson-Laird [1] defined a mental model as an internal representation of a situation that preserves its structure and can be used to simulate, predict, and

reason about it. This definition focuses on the *representational* function of mental models — how they encode external reality internally.

The management literature, particularly influenced by Charlie Munger [2], uses “mental model” in a broader sense: any cognitive framework, heuristic, or reasoning pattern that helps a decision-maker interpret a problem and choose a response. This includes not only representational models (e.g., stock-and-flow diagrams) but also transformational frameworks (e.g., inversion, first-principles reasoning, pre-mortem analysis). Base120 adopts this broader definition and formalizes it as a set of named, versioned reasoning operators.

2.2 Existing Mental-Models Resources

We surveyed existing mental-models resources on GitHub and the web (August 2026). They fall into three categories:

Content sites. Farnam Street (fs.blog) [3] provides detailed explanations of ~100 mental models with examples and application guidance. The models are not structured into a taxonomy and are not accessible programmatically.

Curated link lists. awesome-concepts [4] (624 GitHub stars) is a curated list of links to mental-models content across the web. awesome-mental-models [7] is a similar list. These provide discovery but no executable tooling.

Platform-specific skill packs. cc-thinking-skills [5] (946 stars) is a Claude Code skill pack with 28 mental models. mental-models-os [8] is a similar Claude Code-centric package with 98 models. These are executable but only within their host platform (Claude Code) and do not provide a general Python library, CLI, or MCP server.

Academic reasoning frameworks. pyreason [9] (344 stars) provides neurosymbolic logic inference. llm-reasoners [10] (2,341 stars) provides LLM reasoning research tooling. These are in a different category — logic inference and LLM training/inference rather than decision-making mental models — but are sometimes confused with mental-models libraries.

No existing resource provides a general-purpose Python library with a structured taxonomy, zero dependencies, multi-surface delivery (Python + CLI + MCP), and an audit trail. Base120 fills this gap.

2.3 The Model Context Protocol

The Model Context Protocol (MCP) [11] is an open standard for connecting AI agents to external tools and data sources. MCP-compatible agents (including Claude Code, Cursor, Codex, and Copilot) can invoke MCP server tools to access external capabilities. Base120 ships with an MCP server entry point (`base120-mcp`), allowing any MCP-compatible agent to use the 120 operators as reasoning tools.

3. The Six-Family Taxonomy

3.1 Design Rationale

Base120 organizes its 120 operators into six transformation families based on the type of cognitive transformation each operator performs. The families are:

Family	Code	Transformation	Question it answers	Operators
Perspective	P	Changes the viewpoint	“How else can I see this?”	18
Inversion	IN	Reverses the problem	“What if the opposite is true?”	18
Composition	CO	Assembles from parts	“How do I build this from components?”	20
Decomposition	DE	Breaks into parts	“What are the pieces?”	20
Recursion	RE	Applies self-reference	“How does this feed back on itself?”	20
Systems	SY	Analyzes whole-system dynamics	“How does the whole behave?”	20

The six families are derived from three orthogonal axes of cognitive transformation:

1. **Direction:** Perspective and Inversion are *lateral* transformations (changing the angle of view). Composition and Decomposition are *structural* transformations (changing the granularity). Recursion and Systems are *dynamic* transformations (changing the temporal or causal structure).
2. **Polarity:** Each axis has a complementary pair. Perspective (multiple views) $\leftrightarrow$ Inversion (reverse view). Composition (assemble) $\leftrightarrow$ Decomposition (disassemble). Recursion (self-reference) $\leftrightarrow$ Systems (emergent wholeness).
3. **Scale:** Perspective/Inversion operate at the *conceptual* scale. Composition/Decomposition operate at the *structural* scale. Recursion/Systems operate at the *dynamic* scale.

This three-axis structure ensures that the taxonomy covers the full space of cognitive transformations without overlap. Every reasoning operator can be classified by its direction (lateral/structural/dynamic), polarity (positive/negative), and scale (conceptual/structural/dynamic).

3.2 Family Descriptions

Perspective (P1–P18). Operators that change the viewpoint from which a problem is examined. Includes First Principles Framing (P1), Empathy Mapping

(P5), Point-of-View Anchoring (P6), Context Windowing (P10), and Horizon Scanning (P18). Perspective operators answer: “How else can I see this?”

Inversion (IN1–IN18). Operators that reverse the problem or examine its negation. Includes Reductio ad Absurdum (IN1), Worst-Case Analysis (IN5), Pre-Mortem (IN6), and Regret Minimization (IN7). Inversion operators answer: “What if the opposite is true?”

Composition (CO1–CO20). Operators that assemble solutions from components. Includes Modularity (CO1), Interface Design (CO4), Protocol Layering (CO5), and Saga Pattern (CO17). Composition operators answer: “How do I build this from parts?”

Decomposition (DE1–DE20). Operators that break problems into parts. Includes Root Cause Analysis (DE1), Five Whys (DE2), Separation of Concerns (DE5), and Domain-Driven Design (DE10). Decomposition operators answer: “What are the pieces?”

Recursion (RE1–RE20). Operators that apply self-reference or iteration. Includes Feedback Loop (RE1), Recursion (RE2), Meta-Reasoning (RE5), and Reinforcement Learning (RE15). Recursion operators answer: “How does this feed back on itself?”

Systems (SY1–SY20). Operators that analyze whole-system dynamics. Includes Causal Loop Diagrams (SY1), Leverage Points (SY4), Reinforcing Feedback (SY13), and Antifragility (SY16). Systems operators answer: “How does the whole behave?”

3.3 Composition Grammar

The six families chain naturally in structured decision-making arcs. A common arc is:

Perspective -> Inversion -> Decomposition -> Composition -> Recursion -> Systems

This arc corresponds to a complete reasoning cycle: 1. **Perspective:** Gather multiple viewpoints on the problem. 2. **Inversion:** Stress-test by examining what would make it fail. 3. **Decomposition:** Break the problem into solvable parts. 4. **Composition:** Assemble solutions for each part into a whole. 5. **Recursion:** Check for feedback loops and self-reference in the solution. 6. **Systems:** Analyze how the solution behaves as a system.

Not every decision requires all six families, but the taxonomy provides a completeness check: if a decision-making process has not considered at least one operator from each family, there is a mode of thinking that has not been exercised.

4. The Canonical Registry

The canonical registry is a single YAML file (`Base120_Canonical_Model_Registry.yaml`) that is the source of truth for all 120 operators. The registry is versioned (currently v1.0.0) and frozen — changes require a SemVer bump.

4.1 Registry Format

```
schema_version: base120.schema.v1
canonical: true
version: "1.0.0"
models:
  - id: P1
    name: First Principles Framing
    family: P
    definition: >-
      Reduce complex problems to foundational truths
      that cannot be further simplified
  - id: P2
    name: Stakeholder Mapping
    family: P
    definition: >-
      Identify all parties with interest, influence,
      or impact in a system or decision
# ... 118 more
```

Each operator has: - **id**: A unique identifier (family code + number, e.g., P1, IN6, SY13) - **name**: A human-readable name - **family**: The transformation family (P, IN, CO, DE, RE, or SY) - **definition**: A precise definition of what the operator does

4.2 Cross-Language Consumption

The YAML registry is language-agnostic. Implementations in any language can read it with any YAML parser:

```
import yaml
with open("Base120_Canonical_Model_Registry.yaml") as f:
    registry = yaml.safe_load(f)
models = {m["id"]: m for m in registry["models"]}

import yaml from 'js-yaml';
const registry = yaml.load(readFileSync('Base120_Canonical_Model_Registry.yaml', 'utf8'));
```

This design ensures that Base120 is not coupled to Python. Other-language implementations (Rust, Go, TypeScript) conform to the frozen registry and can be validated against the test corpus in `tests/`.

4.3 Versioning Policy

The registry follows a frozen-canon policy: - **v1.0.0**: The current canonical version. 120 operators across 6 families. - **Permitted changes**: Security fixes, CI hardening, documentation, corpus additions. - **Prohibited changes**: Schema changes, registry modifications, breaking changes. - **Future versions**: New operators may be added in assessment queues for future versions, but v1.0.0 remains frozen.

5. The Python SDK

5.1 Design Principles

The Python SDK is designed around four principles:

1. **Stdlib-only**: Zero third-party runtime dependencies. The entire library runs on Python 3.11+ with no installs beyond `pip install base120`.
2. **Deterministic**: Same input, same output. No LLM calls, no network, no randomness.
3. **Tuple-native**: Every operator application produces a JSONL tuple for ledger persistence.
4. **Agent-friendly**: Works with any MCP-compatible agent via the `base120-mcp` entry point.

5.2 Engine API

```
from base120 import Engine, Ledger

engine = Engine()

# Look up an operator by ID
operator = engine.get("P6")
print(operator.name)          # -> Point-of-View Anchoring
print(operator.family)        # -> P (Perspective)

# Generate an operator-specific prompt for a problem
prompt = engine.prompt("P6", "How should we price the certification tier?")

# Apply an operator and produce a ledger tuple
result = engine.record(
    "P6",
    "How should we price the certification tier?",
    "Anchor the offer to the compliance officer's risk budget.",
    0.85, # confidence score
)
```

The `Engine` class provides four methods: `get` (lookup), `list` (all operators), `families` (family metadata), and `prompt` (generate an operator-specific prompt for a problem). The `record` method applies an operator and produces a `Result` object whose `to_tuple()` method returns a JSONL-serializable tuple.

5.3 CLI

```
base120 list           # List all 120 operators
base120 get P6         # Inspect one operator
base120 prompt P6 "..."# Generate an operator-specific prompt
base120 families       # List the 6 transformation families
```

5.4 MCP Server

```
base120-mcp           # Run the MCP server
```

Once running, any MCP-compatible agent can call `base120.get`, `base120.list`, `base120.prompt`, and `base120.families` as MCP tools. Configuration for Claude Code and Cursor is documented in the README.

6. The Append-Only Ledger

Every operator application can be persisted as a JSONL tuple to an append-only ledger:

```
from base120 import Engine, Ledger

engine = Engine()
ledger = Ledger("decisions.jsonl")

result = engine.record(
    "DE1",           # operator ID
    "Reduce release risk.", # problem
    "Split blockers by owner.", # response
    0.9,            # confidence
)

ledger.append(result.to_tuple())

# Query high-drift records (confidence < threshold)
high_drift = ledger.cut(0.5)
```

6.1 Tuple Format

Each ledger entry is a JSONL tuple with: - `operator_id`: The operator code (e.g., "DE1") - `problem`: The problem statement - `response`: The applied response -

confidence: Float 0.0–1.0 - timestamp: ISO 8601 timestamp

6.2 Governance Properties

The ledger is append-only — records are never modified or deleted. This provides three governance properties:

1. **Retrospective verifiability:** Any decision can be reconstructed from the ledger.
2. **Tamper-evidence:** Modifications are detectable (records are immutable).
3. **Auditability:** The ledger can be reviewed by a governance process to assess decision quality.

The ledger connects Base120 to the broader HUMMBL governance architecture [6], where it provides the reasoning-operator layer of the governance tuple — the artifact that records *which reasoning was applied* to a given decision.

7. Use Cases

7.1 Structured Human Decision-Making

A human decision-maker can use Base120 to structure a complex decision by applying operators from multiple families in sequence:

Problem: “Should we migrate from REST to GraphQL?”

Step 1 -- P1 (First Principles): What are the irreducible requirements?
Step 2 -- IN5 (Worst-Case Analysis): What if the migration takes 6 months?
Step 3 -- DE5 (Separation of Concerns): Which parts need flexibility?
Step 4 -- C01 (Modularity): Can we support both during transition?
Step 5 -- SY13 (Feedback Loops): How do we know it's working?

Each step names the model, applies it, and passes output to the next. The ledger records the full reasoning chain for retrospective review.

7.2 AI Agent Reasoning via MCP

An MCP-compatible agent (Claude Code, Cursor, Codex) can use Base120 operators as reasoning tools:

```
# Agent calls base120.prompt via MCP
prompt = mcp_call("base120.prompt", {
    "operator_id": "IN6",
    "problem": "Should we launch the new pricing tier next Monday?"
})
# Agent receives a pre-mortem prompt and applies it
```

This gives agents a structured vocabulary of 120 reasoning primitives, improving the quality of agent decision-making by ensuring that multiple modes of thinking are exercised.

7.3 Multi-Agent Coordination with Audit Trails

In a multi-agent system, different agents can apply different families of operators, with the ledger recording who applied what:

```
# Agent A applies Perspective operators
for op_id in ["P1", "P5", "P6", "P10"]:
    result = engine.record(op_id, "Design a rate limiter", response_a, 0.8)
    ledger.append(result.to_tuple())

# Agent B applies Inversion operators
for op_id in ["IN5", "IN6", "IN15"]:
    result = engine.record(op_id, "Design a rate limiter", response_b, 0.7)
    ledger.append(result.to_tuple())

# Agent C applies Systems operators
for op_id in ["SY1", "SY12", "SY13"]:
    result = engine.record(op_id, "Design a rate limiter", response_c, 0.9)
    ledger.append(result.to_tuple())
```

The ledger provides a complete audit trail of which agent applied which reasoning operator, with confidence scores for each application. This is the foundation for governance-readable multi-agent decision records.

8. Comparison with Existing Resources

Feature	Base120	cc-thinking-skills [5]	mental-models-os [8]	awesome- concepts [4]
Model count	120	28	98	~100 (links)
Taxonomy	6 trans- forma- tion fami- lies	Flat list	Flat list	Flat list
Python library	Yes	No (JS skills)	Yes	No
CLI	Yes	No	Yes	No
MCP server	Yes	No	Yes	No

Feature	Base120	cc-thinking-skills [5]	mental-models-os [8]	awesome- concepts [4]
Dependencies	Zero (stdlib only)	N/A	Standard	N/A
Ledger	Yes (append- only JSONL)	No	No	No
License	Apache 2.0	MIT	MIT	CC0
Platform	Any Python 3.11+	Claude Code only	Python + Claude Code	Web (links)
Deterministic	Yes	No (LLM-based)	No	N/A
Cross- language reg- istry	Yes (YAML)	No	No	N/A

Base120 is the only resource that provides all of: a structured taxonomy, a general Python library, a CLI, an MCP server, zero dependencies, an audit trail, and a cross-language registry.

9. Limitations and Future Work

9.1 Limitations

- **English-only:** Operator names and definitions are in English. Translations are a future contribution opportunity.
- **No LLM integration:** Base120 provides reasoning structure, not LLM inference. Users must supply the intelligence (human or AI) to apply each operator.
- **No automated operator selection:** Base120 does not recommend which operator to apply for a given problem. Operator selection is a human or agent decision.
- **Software-engineering bias in CO/DE:** The Composition and Decomposition families include software-architecture patterns (e.g., Saga, CQRS, Event Sourcing) alongside general-purpose decomposition operators. This reflects the library’s origin in a software-engineering context.

9.2 Future Work

- **Other-language implementations:** Rust, Go, TypeScript ports conforming to the frozen registry.
 - **Operator recommendation:** A retrieval layer that suggests operators based on problem characteristics.
 - **Evaluation framework:** Empirical evaluation of whether structured operator application improves decision quality compared to unstructured reasoning.
 - **Domain-specific operator packs:** Curated subsets of operators for specific domains (product, engineering, policy, research).
-

10. Conclusion

Base120 is a canonical registry of 120 reasoning operators organized into six transformation families, with a stdlib-only Python SDK, CLI, MCP server, and append-only ledger. It fills a structural gap in the mental-models landscape: no existing resource provides a general-purpose Python library with a structured taxonomy, zero dependencies, multi-surface delivery, and an audit trail. The six-family taxonomy provides a navigation map, completeness check, and composition grammar for structured reasoning. The canonical YAML registry enables cross-language implementations. The ledger enables governance-readable decision records. Base120 is deployed in production as the cognitive layer of the HUMMBL multi-agent architecture.

References

- [1] Johnson-Laird, P. N. (1983). *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press.
- [2] Munger, C. T. (1994). *A Lesson on Elementary Worldly Wisdom*. USC Business School commencement remarks. Widely circulated; see also *Poor Charlie's Almanack* (2005).
- [3] Farnam Street. “Mental Models.” <https://fs.blog/mental-models/>
- [4] Madon, L. “awesome-concepts.” <https://github.com/lukasz-madon/awesome-concepts>
- [5] Boudreaux, T. “cc-thinking-skills.” <https://github.com/tjboudreaux/cc-thinking-skills>
- [6] HUMMBL. “HUMMBL Cognitive AI Architecture.” <https://hummb1.io>
- [7] Lemaire, A. “awesome-mental-models.” <https://github.com/AdrienLemaire/awesome-mental-models>

- [8] Arnoux, D. “mental-models-os.” <https://github.com/darnoux/mental-models-os>
- [9] lab-v2. “pyreason.” <https://github.com/lab-v2/pyreason>
- [10] maitrix-org. “llm-reasoners.” <https://github.com/maitrix-org/llm-reasoners>
- [11] Anthropic. “Model Context Protocol.” <https://modelcontextprotocol.io/>
-

Software Availability

Repository: <https://github.com/hummb1-io/base120> **PyPI:** <https://pypi.org/project/base120/>
License: Apache 2.0 **Python version:** 3.11+ **Dependencies:** Zero
(stdlib only) **Test suite:** 120 tests, all passing **Canonical registry:**
`Base120_Canonical_Model_Registry.yaml` (v1.0.0, frozen)