Do we delegate control flow to the AI or do we specify control flow at the outset programmatically to control the behaviour of agents? Can also specify control flow using AI programmatically? \*Pydantic has control of

Scrape Docs for:

1. Autogen
2. Dspy
3. ~~Semantic kernel~~
4. Google OR
5. Z3
6. Powershell
7. Pynusmv
8. Langchain
9. Langraph
10. CrewAI
11. Crawl4AI
12. ~~LlamaIndex~~
13. Smolagents
14. Pydantic AI
15. ~~Mastra?~~
16. ~~Llamaindex.~~
17. ~~OpenAI evals~~
18. <https://github.com/AlgebraicJulia/Catlab.jl>
19. sympy/sagemath
20. ~~Lean bindings?~~

Expose tools for:

Workflows -

1. **Meta Prompting**: Request the AI to rewrite or refine your original prompt before generating an answer.
2. **Chain-of-Thought**: Instruct the AI to break down its reasoning process step-by-step before producing an output or recommendation
3. **Tree-of-Thought**: Enable the AI to explore multiple reasoning paths simultaneously, evaluating different approaches before selecting the optimal solution (this was the missing piece many of you mentioned).
4. **Prompt Chaining**: Link multiple prompts together, where each output becomes the input for the next task, forming a structured flow that simulates layered human thinking.
5. **Generate Knowledge**: Ask the AI to explain frameworks, techniques, or concepts using structured steps, clear definitions, and practical examples.
6. **Retrieval-Augmented Generation (RAG)**: Enables AI to perform live internet searches and combine external data with its reasoning.
7. **Reflexion**: The AI critiques its own response for flaws and improves it based on that analysis.
8. **ReAct**: Ask the AI to plan out how it will solve the task (reasoning), perform required steps (actions), and then deliver a final, clear result

These tools should be able to cover 80-90% of the most common AI workloads.

Research with crawl4AI?

General research tool (perplexity, openAI research preview gpt4, any number of free tools? searXNG?)

Docs indexing and context management for chat with personal docs.

Hierarchical planning and decomposition with research using my custom agent (ROMA??)

Build program logic with dspy and use langchain to impart with memory for low level custom workflows i.e. programming with LLMs.

Memory MDP using memento.

Database Postgresql for large scale long context memory? 4x types (relational, semantic, procedural and episodic)

Taskmaster - large context tasks + langgraph.

Thought as code using smolagents.

Category theory for abstract understanding (<https://github.com/AlgebraicJulia/Catlab.jl> + <https://github.com/JuliaPy/PythonCall.jl> )

Math tools - sympy/sagemath, z3, googleOR

Model checking - pynusmv.

Debugging.

File manipulation/shell scripting.

API/MCP connections.

Addition of tools.

Start/stop/handoff/guardrail/resource constraints + call agent using crewAI or autogen or agents sdk with or without tools using semantic kernel.

Experta for expert system.

RQ for workers.

AutoML for perceptual tasks or statistical inference.

Knowledge and heuristic discovery with eurisko, data mining and pattern identification using ML.

Execute code in sandbox.

Computer use agent (playwrightMCP + openCUA?)

Spec kit for coding specs.

Markitdown for markdown + OCR + formula recognition/diagram understanding.

Langextract for structured text extraction.

Activepieces for business automation.

Evals for evaluating system performance/metric telemetry.

Antifragility MCP (to ensure more than robustness but also boosting with feedback - this can just be some kind of boosting library)

Creative arts generative AI.

* **XGBoost (Extreme Gradient Boosting):**This is a highly optimized and widely used library for gradient boosting. It is known for its speed, accuracy, and ability to handle various data types, including tabular data. XGBoost is particularly effective in competitive machine learning scenarios and for achieving high performance.
* **LightGBM (Light Gradient Boosting Machine):**Developed by Microsoft, LightGBM is another powerful gradient boosting framework that prioritizes speed and efficiency. It is often faster than XGBoost, especially with large datasets, due to its optimized algorithms and techniques like GOSS (Gradient-based One-Side Sampling) and EFB (Exclusive Feature Bundling).
* **CatBoost:**Developed by Yandex, CatBoost is designed to handle categorical features effectively, automatically converting them into numerical representations without requiring extensive preprocessing. It is known for its robustness and good performance, especially when dealing with datasets containing many categorical variables.

Everything passes through litellm proxy for token tracking + telemetry. Everything is checked with QA/governance/safety/ethics agent.

Design UI - I think I have sketched this out under customer profile?

Chat agent and worker agent have telemetry - these are both in the main tab - running execution string displays workers actions over time summary. Play and pause on both with guardrails and constraints. Emission of tasks - can work remotely by email or directly in app. Assets are files or actions completed by system. Can view goal hierarchy and current subgoal + dependencies as graph in another tab. Can see progress and milestones in telemetry.

5 tabs - chat, workers, tasks for review, goal trees, telemetry.

Settings separately.

Previous thoughts:

The chat agent triages into one of the 5 main workflows (plus a mentoring/teaching mode). The worker agent has a start pause and resume and stop. Pause just gets the agent to stop where it is and persist state but stop actually causes agent to start from scratch using existing progress so far.

Exhibit status updates based on progress bar (with additional info) + telemetry through liteLLM with graphana and prometheus monitoring via openwebUI backend just plug into their tools. Can plug this into the demo + can only work on one project at a time. Just use mastra and keep all code in typescript for demo agent but have this obfuscated?? Have a task list for human to complete and emit a worker status. Can export project as zip folder. Can also view project assets which are stored locally only during session active (session must be kept active - after 24 hrs connection will close). Add tools directly to openwebUI and plug into their backend. Use their model providers drop down as well. Stream via sse/websockets. On the backend just expose workflows and agent frameworks with hierarchical task decomposition.

Ask 4us, pain point, tedious automation, AI coworker to help you as example prompts.

—-----------------------------------

Great — this is an exciting, ambitious stack. Short answer first: **yes** — combining an LLM with (1) optimization (OR-Tools), (2) SMT solving (Z3), (3) model checking (pynusmv), (4) symbolic/math CAS (SageMath), and (5) category-theory tooling (Catlab.jl) can *substantially* increase an LLM’s ability to do rigorous, high-complexity problem solving — provided you design a disciplined orchestration layer that translates between the LLM’s natural-language reasoning and the formal solvers’ languages and accepts that each tool has different guarantees (optimality, soundness, decidability, performance). Below I give a pragmatic architecture, integration patterns, end-to-end workflows, pitfalls, and an incremental build plan you can action now.

# **1) High-level architecture (conceptual)**

Think of the system as **orchestrator + tool adapters + LLM** with supporting subsystems (logging, sandboxing, caching, scheduler, verifier).

Main components:

1. **Orchestrator / Controller**
   * Receives the user problem (NL + optional formal spec).
   * Manages the LLM as a planner/translator and coordinates solver calls.
   * Maintains job state, retries, resource limits and a provenance ledger (what was asked / inputs / outputs / who invoked what step).
   * Implements strategies: sequential refinement, parallel solve, speculative calls.
2. **LLM agent**
   * Roles: *Planner* (decompose the problem), *Translator* (emit formal queries), *Integrator* (combine solver outputs), *Explainer* (generate human-readable reports).
   * Not a blind "single-step" caller: use few-shot templates + schema enforcement so outputs are parseable.
3. **Tool Adapters (microservices or libraries)**
   * **OR-Tools adapter**: receives optimization problem (e.g., routing + constraints) in a canonical JSON model and returns solution + objective + diagnostics.
   * **Z3 adapter**: accepts SMT-LIB or high-level AST, returns sat/unsat, model, proofs (when available).
   * **pynusmv adapter**: accepts transition system + LTL/CTL properties, returns counterexamples/traces or verified.
   * **SageMath adapter**: symbolic math, algebraic simplification, number theory, proof search helpers.
   * **Catlab.jl adapter**: category-theory transformations, diagram rewriting, algebraic specifications — exposed via a JSON/RPC boundary (PyJulia or a small HTTP/GRPC microservice).
   * Each adapter hides implementation details and translates between a single internal canonical representation and the tool’s native format.
4. **Verifier / Validator**
   * Re-checks LLM translations (sanity checks), validates solver outputs, and runs cross-tool consistency checks (e.g., verify OR-Tools route satisfies Z3 constraints).
5. **Provenance + Audit store**
   * Stores inputs/outputs, solver versions, timeouts, counterexamples, and final conclusions for reproducibility.
6. **Resource manager & sandbox**
   * Limits CPU/memory, imposes timeouts on solvers, and sandboxes Julia/Python/Sage processes to prevent runaway compute and to secure system calls.

# **2) Data flow & interaction patterns**

There are a few canonical patterns for mixing LLM reasoning and formal tools.

A. **Planner → Formalize → Solve → Verify → Explain**

1. LLM decomposes the problem and proposes formal models (JSON ASTs).
2. Orchestrator sends model to appropriate adapters (OR-Tools / Z3 / Sage).
3. Adapters return results; orchestrator asks LLM to interpret and synthesize.
4. Verifier cross-checks outputs with independent solver or encoding.

B. **Counterexample-guided refinement (CEGAR style)**

* LLM produces candidate solution.
* Z3/pynusmv tries to find counterexamples; if found, counterexample is returned to LLM which refines the model.
* Iterate until no counterexample within resource limits.

C. **Hybrid co-solve (opt + SMT)**

* OR-Tools gives near-optimal solution for numeric/NP problems; Z3 verifies logical consistency and enforces discrete invariants. If Z3 finds inconsistency, constraints are tightened and OR-Tools re-solves.

D. **Symbolic algebra + categorical rewriting**

* Use Catlab to perform high-level algebraic rewrites, produce a canonical form, then hand it to Sage for simplification or to Z3 for decidability checks.

# **3) Representation: canonical intermediate languages**

A single canonical representation avoids repeated ad-hoc translation. Examples:

**Constraint JSON (for optimization/SMT)** {

"vars": [{"name":"x1","type":"int","lb":0,"ub":10}, ...],

"constraints":[{"type":"linear","expr":"2\*x1 + 3\*x2 <= 7"}, {"type":"logic","expr":"implies(p,q)"}],

"objective":{"sense":"min","expr":"sum(cost[i]\*x[i])"}

}

**Transition system** for model checking:  
  
 { "states": [...], "init": [...], "trans": [{"from":"s","to":"t","guard":"g", "action":"a"}], "props":[{"type":"LTL","formula":"G(!fail)"}] }

* **Category AST** for Catlab: nodes/objects, morphisms/arrows, diagrams as edge lists.

Use typed schemas (JSON Schema) so the LLM must produce validated structured outputs.

# **4) Practical integration techniques**

### **Calling Catlab.jl from Python**

* **Two safe options**:  
  1. **PyJulia** — embed Julia inside Python. Good when latency allowed and you can manage the Julia runtime in the same process.
  2. **Microservice** — run a small Julia HTTP/GRPC server that exposes operations (rewrite, compose, canonicalize) returning JSON. This isolates Julia and is easier to scale.

### **Z3**

* Use Z3’s Python bindings or SMT-LIB text. Provide the LLM with templates for common encoding patterns to avoid buggy translations (e.g., how to encode arrays/integers/bitvectors).

### **OR-Tools**

* Construct models via OR-Tools Python API inside the adapter. For complex objectives (multi-objective, lexicographic), translate LLM intents into solver configuration.

### **SageMath**

* Run Sage either via a subprocess or via sagecell/Jupyter kernel. For low latency, create a persistent Sage process with an RPC interface.

### **pynusmv**

* Build transition systems programmatically from the canonical representation and allow the LLM to propose properties in temporal logic templates (LTL/CTL). Provide examples.

# **5) LLM prompting & tooling contract**

* Use **structured prompts** (system + few-shot) that require the LLM produce machine-parsable outputs (JSON) only. Example:  
  + System prompt: “You are a translator to SMT/OPT/TRANSITION SYSTEM. Output must be valid JSON matching schema X.”
  + Few shot: show 3 examples: NL → JSON (for the target solver).
* Validate LLM outputs with JSON Schema before sending to adapters. If invalid, either auto-fix via deterministic rules or request a corrected translation (but prefer automatic fixes for robustness).

# **6) Combining outputs & conflict resolution**

* **Priority rules**: declarative invariants shall be enforced by Z3 (soundness), optimization objectives by OR-Tools (optimality). If they conflict, treat Z3 invariants as higher priority and re-solve optimization with stronger constraints.
* **Voting/ensembling**: for uncertain translations, call multiple prompt templates + majority merge or merge via proof obligations.
* **Counterexample injection**: when model checking produces counterexample traces, feed them back to the LLM for repair.

# **7) Performance & resource management**

* **Timeboxing**: assign time budgets per solver invocation.
* **Parallelization**: run different solver encodings in parallel (different encodings often yield very different performance).
* **Caching**: cache solver results keyed by canonical representation hash.
* **Asynchronous orchestration**: orchestrator should be asynchronous (coroutines, message queue), but note the system must immediately return a result in each request — do not promise future-only results.

# **8) Safety, reproducibility, and explainability**

* Always store solver versions, random seeds, and the full JSON exchange.
* Keep deterministic seeds for solvers if you want reproducible runs.
* Use Z3 proofs (where supported) to produce machine-checkable evidence — these can be inspected by the LLM to produce human explanations.

# **9) Example end-to-end workflow (vehicle routing + safety constraints)**

1. User: “Find routes for 50 deliveries with priority constraints and fragile items that must not be on the same truck as hazardous items.”
2. LLM planner: decomposes into routing + assignment + forbidden co-locations + capacity constraints → emits canonical JSON.
3. OR-Tools adapter: solves VRP for routes minimizing distance; returns routes, load, objective.
4. Z3 adapter: verifies logical invariants (e.g., “no truck has both fragile and hazardous”) on the OR-Tools result; if SAT (i.e., invariant violated), it returns counterexample assignments.
5. Orchestrator: if Z3 produced counterexample, it triggers an incremental re-encode: add a stronger constraint and re-run OR-Tools.
6. After stable solution, pynusmv can be used to verify dynamic properties like “if truck A breaks down, can remaining trucks still cover all deliveries within time windows?” If not, produce contingency plan.
7. Catlab.jl used to reason about compositional properties (e.g., compose schedules and resource flows) and detect architectural-level inconsistencies.
8. LLM synthesizes final human-readable plan + formal proof/sketch of constraints satisfied.

# **10) Implementation sketch (adapter API & prompt pattern)**

Suggested simple adapter interface (Python pseudo):

class ToolAdapter:

def solve(self, model\_json: dict, timeout\_s: int) -> dict:

"""Take canonical model, return {'status':'ok','result':...,'diagnostics':...}"""

def verify(self, model\_json: dict, candidate: dict, timeout\_s: int) -> dict:

"""Ask tool to check candidate solution against invariants."""

LLM prompt pattern (schematic):

SYSTEM: You are a translator. Output EXACT JSON that matches the schema. Do NOT output extra text.

USER: Problem: [Natural description...]

EXAMPLES:

- NL -> JSON example 1

- NL -> JSON example 2

# **11) Testing & evaluation**

* Start with **unit tests** for each adapter: known inputs → expected outputs.
* **Integration tests**: small toy problems where you can verify result by hand.
* **Fidelity tests**: compare LLM translations to hand-written encodings; measure parser error rate.
* **Robustness**: fuzz LLM prompts; measure solver failures and timeouts.
* **Metrics**: success rate (problem solved), time to solution, number of solver iteration cycles, number of counterexamples, human evaluation of explanations.

# **12) Staged rollout plan (practical)**

1. **Prototype**: LLM + OR-Tools only. Build canonical JSON, LLM translator, OR-Tools adapter, basic verification.
2. **Add Z3**: integrate SMT checks for invariants and counterexample loop.
3. **Add Sage**: hook in for math/symbolic tasks where simplification or exact algebra is required.
4. **Add pynusmv**: encode dynamic/temporal aspects and verify high-level properties.
5. **Add Catlab.jl**: as microservice for algebraic/categorical reasoning — use it where compositionality matters.
6. **Hardening**: sandbox, logging, monitoring, metrics, UI for human-in-the-loop fixes.
7. **Scale**: containerize adapters, orchestrate via Kubernetes or similar, add autoscaling.

# **13) Practical pitfalls & how to mitigate them**

* **LLM hallucination in formal encodings**: mitigate via strict schema validation + small unit tests + ask LLM to include concrete examples.
* **Mismatch of guarantees**: optimization gives heuristics; SMT gives soundness for decidable fragments. Explicitly mark which results are formal proofs vs heuristics.
* **Performance blowups**: use timeouts, fallback encodings, and simpler abstractions.
* **Interoperability friction (Julia ↔ Python)**: prefer microservice boundary to avoid runtime conflicts, or use PyJulia carefully with isolated processes.
* **Undecidability / scale**: when your problem is undecidable or huge, prefer approximation layers (relax discrete constraints, hierarchical solving).

# **14) UX: how to present results to users**

* Show: final plan + “confidence” (solver types used, proof/counterexample present, number of iterations).
* Provide an interactive trace: allow a user to ask “why” and show the chain: NL → JSON → solver call → solver output → counterexample → revision.
* Offer toggles: prefer soundness (enforce Z3 invariants at all cost) vs. prefer performance/near-optimality (allow slight relaxations).

# **15) Example use-case mapping**

* **Logistics optimization**: OR-Tools + Z3 + model checking for contingencies.
* **Synthesis + verification**: propose design via LLM, use Catlab to check compositional laws, Z3 to ensure type/invariant correctness, pynusmv for temporal safety.
* **Math research assistant**: LLM suggests proof steps, Sage verifies algebraic manipulations, Z3 tries to automatically discharge proof obligations.

If you want, I can:

* sketch a concrete JSON schema for the canonical constraint representation (very useful to lock down early), or
* produce a minimal prototype orchestration in Python that demonstrates LLM→OR-Tools→Z3 loop on a small toy problem (vehicle routing or scheduling).

Which of those would you like to see first?

—---------------------------------------

### Memory

- Docs indexing and context management for chat with personal docs using vector DB

- Memory MDP using memento

- Database Postgresql for long-context memory (relational, semantic, procedural, episodic)

- Regex/Grep

- Dependency analysis.

- Advanced text compression methods.

### Reasoning

- Taskmaster (large-context tasks + LangGraph)

- Thought-as-code (smolagents)

- Hierarchical planning and decomposition with research (ROMA)

- Build program logic with dspy and pydantic (memory) for low-level workflows

- Spec kit for coding specs

- Category theory tools (Catlab.jl + PythonCall)

- Math tools (SymPy/SageMath, Z3, Google OR-Tools)

- Model checking (PyNuSMV)

- Experta (expert system)

- Creative arts generative AI

### Learning

- Research with crawl4AI

- General research tools (Perplexity, OpenAI Research, SearXNG, etc.)

- Tool for upskilling in a domain / creating new tools

- AutoML for perceptual/statistical tasks

- Knowledge/heuristic discovery (Eurisko), data mining, pattern identification via ML

- Markitdown (markdown + OCR + formula/diagram understanding)

- Langextract (structured text extraction)

- Antifragility MCP (feedback-based boosting)

### Agentic/System (doesn’t naturally fit memory/reasoning/learning)

- Debugging/programming tools

- File manipulation/shell scripting

- API/MCP connections

- Start/stop/handoff/guardrails/resource constraints; call agents (Autogen, Agents SDK, PydanticAI)

- RQ (workers)

- Execute code in sandbox

- Computer-use agent (Playwright MCP, OpenCUA)

- Activepieces (business automation)

- LiteLLM for evaluating system performance/telemetry

- Homoiconic tool (tool for acting on/modifying self).

—-----------------------------------Tool Hashes—--------------------------------

I'll analyze each tool based on the 5-axis decomposition from the strategy planner and assign numerical values (0-3) for each axis: R\_c (compressibility), R\_s (structure), S (semantics), M (maturity), D (determinism), and E (epistemic grounding).

[1 tool called]

Let me analyze each tool based on the 5-axis framework from the strategy planner. I'll examine the algorithm's scoring criteria and apply them systematically.

[3 tools called]

Now I have the complete rubrics from the strategy planner. Let me analyze each tool systematically:

\*\*5-Axis Scoring Criteria:\*\*

- \*\*R\_c (Compressibility)\*\*: 3 if CR ≥ 2.00; 2 if 1.40 ≤ CR < 2.00; 1 if 1.10 ≤ CR < 1.40; 0 if CR < 1.10

- \*\*R\_s (Structure)\*\*: 3 if preserve ≥ 80% key fields/relations and ≥ 3 critical invariants; 2 if 50-79% and 1-2 invariants; 1 if ad-hoc structure helps; 0 if none

- \*\*S (Semantics)\*\*: 0 extensional (finite/exhaustive by examples); 1 large-but-enumerable; 2 rules/constraints dominate; 3 intensional (types essential)

- \*\*M (Maturity)\*\*: 0 open; 1 heuristics exist; 2 standard algorithms with partial proofs/benchmarks; 3 solved with strong libraries and consensus

- \*\*D (Determinism)\*\*: 0 irreducible/non-stationary; 1 weak structure; 2 stable after transforms; 3 reducible with parsimonious model and R² ≥ 0.80

- \*\*E (Epistemic)\*\*: 0 a posteriori (must cite); 1 mixed; 2 largely derivable; 3 a priori (proofs/model checking feasible)

[1 tool called]

## Tool Analysis with 5-Axis Scores

### Memory Tools

\*\*1. Docs indexing and context management (Vector DB)\*\*

- \*\*R\_c: 3\*\* - High compression ratio through embeddings and indexing

- \*\*R\_s: 2\*\* - Moderate structure preservation (metadata, hierarchies)

- \*\*S: 1\*\* - Large enumerable corpus with similarity search

- \*\*M: 3\*\* - Mature field with established libraries (FAISS, Pinecone, etc.)

- \*\*D: 2\*\* - Stable retrieval patterns, predictable similarity metrics

- \*\*E: 0\*\* - A posteriori knowledge requiring citations

- \*\*Hash: [3, 2, 1, 3, 2, 0]\*\*

\*\*2. Memory MDP using Memento\*\*

- \*\*R\_c: 1\*\* - Low compression (state transitions are sparse)

- \*\*R\_s: 3\*\* - Critical structure preservation (states, transitions, rewards)

- \*\*S: 2\*\* - Rule-based transitions and constraints dominate

- \*\*M: 2\*\* - Standard RL algorithms, partial theoretical grounding

- \*\*D: 2\*\* - Markovian assumptions provide stability

- \*\*E: 2\*\* - Largely derivable from MDP theory

- \*\*Hash: [1, 3, 2, 2, 2, 2]\*\*

\*\*3. Database PostgreSQL (relational, semantic, procedural, episodic)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through normalization

- \*\*R\_s: 3\*\* - Critical structure preservation (schemas, constraints, ACID)

- \*\*S: 2\*\* - SQL rules and constraints dominate

- \*\*M: 3\*\* - Mature with strong consensus and libraries

- \*\*D: 3\*\* - Highly predictable CRUD operations

- \*\*E: 2\*\* - Largely derivable from relational algebra

- \*\*Hash: [2, 3, 2, 3, 3, 2]\*\*

\*\*4. Regex/Grep\*\*

- \*\*R\_c: 3\*\* - High compression through pattern matching

- \*\*R\_s: 1\*\* - Light structure (patterns help but not critical)

- \*\*S: 2\*\* - Pattern rules dominate

- \*\*M: 3\*\* - Solved problem with strong consensus

- \*\*D: 3\*\* - Deterministic pattern matching

- \*\*E: 3\*\* - A priori formal language theory

- \*\*Hash: [3, 1, 2, 3, 3, 3]\*\*

\*\*5. Dependency analysis\*\*

- \*\*R\_c: 2\*\* - Moderate compression (graph representations)

- \*\*R\_s: 3\*\* - Critical graph structure preservation

- \*\*S: 2\*\* - Dependency rules and constraints dominate

- \*\*M: 3\*\* - Well-established graph algorithms

- \*\*D: 3\*\* - Deterministic graph traversal

- \*\*E: 3\*\* - A priori graph theory proofs

- \*\*Hash: [2, 3, 2, 3, 3, 3]\*\*

\*\*6. Advanced text compression methods\*\*

- \*\*R\_c: 3\*\* - Maximum compression by definition

- \*\*R\_s: 1\*\* - Structure helps but not always preserved

- \*\*S: 1\*\* - Large enumerable patterns

- \*\*M: 3\*\* - Mature algorithms (LZ77, Huffman, etc.)

- \*\*D: 3\*\* - Deterministic compression ratios

- \*\*E: 3\*\* - A priori information theory

- \*\*Hash: [3, 1, 1, 3, 3, 3]\*\*

[1 tool called]

### Reasoning Tools

\*\*7. Taskmaster (large-context tasks + LangGraph)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through task decomposition

- \*\*R\_s: 3\*\* - Critical structure (task hierarchies, dependencies)

- \*\*S: 2\*\* - Planning rules and constraints dominate

- \*\*M: 2\*\* - Standard algorithms, emerging best practices

- \*\*D: 2\*\* - Stable after task decomposition transforms

- \*\*E: 1\*\* - Mixed (some derivable planning theory, some empirical)

- \*\*Hash: [2, 3, 2, 2, 2, 1]\*\*

\*\*8. Thought-as-code (smolagents)\*\*

- \*\*R\_c: 1\*\* - Low compression (explicit reasoning traces)

- \*\*R\_s: 2\*\* - Moderate structure (code structure helps)

- \*\*S: 3\*\* - Types and formal reasoning essential

- \*\*M: 1\*\* - Heuristics exist but emerging field

- \*\*D: 1\*\* - Weak structure, context-dependent

- \*\*E: 2\*\* - Largely derivable from formal logic

- \*\*Hash: [1, 2, 3, 1, 1, 2]\*\*

\*\*9. Hierarchical planning and decomposition with research (ROMA)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through hierarchical decomposition

- \*\*R\_s: 3\*\* - Critical hierarchical structure preservation

- \*\*S: 2\*\* - Planning rules and decomposition constraints

- \*\*M: 2\*\* - Standard planning algorithms with research integration

- \*\*D: 2\*\* - Stable hierarchical patterns

- \*\*E: 1\*\* - Mixed (planning theory + empirical research)

- \*\*Hash: [2, 3, 2, 2, 2, 1]\*\*

\*\*10. Build program logic with dspy and pydantic (memory)\*\*

- \*\*R\_c: 1\*\* - Low compression (explicit program structure)

- \*\*R\_s: 3\*\* - Critical type and schema structure

- \*\*S: 3\*\* - Types and formal specifications essential

- \*\*M: 2\*\* - Standard patterns emerging

- \*\*D: 3\*\* - Highly deterministic type checking

- \*\*E: 3\*\* - A priori type theory and formal methods

- \*\*Hash: [1, 3, 3, 2, 3, 3]\*\*

\*\*11. Spec kit for coding specs\*\*

- \*\*R\_c: 2\*\* - Moderate compression through specification patterns

- \*\*R\_s: 3\*\* - Critical specification structure

- \*\*S: 3\*\* - Formal specification types essential

- \*\*M: 2\*\* - Standard specification methods

- \*\*D: 3\*\* - Deterministic specification validation

- \*\*E: 3\*\* - A priori formal specification theory

- \*\*Hash: [2, 3, 3, 2, 3, 3]\*\*

\*\*12. Category theory tools (Catlab.jl + PythonCall)\*\*

- \*\*R\_c: 1\*\* - Low compression (abstract mathematical structures)

- \*\*R\_s: 3\*\* - Critical categorical structure preservation

- \*\*S: 3\*\* - Types and morphisms essential

- \*\*M: 2\*\* - Standard mathematical framework

- \*\*D: 3\*\* - Highly deterministic mathematical operations

- \*\*E: 3\*\* - A priori mathematical proofs

- \*\*Hash: [1, 3, 3, 2, 3, 3]\*\*

\*\*13. Math tools (SymPy/SageMath, Z3, Google OR-Tools)\*\*

- \*\*R\_c: 1\*\* - Low compression (symbolic representations)

- \*\*R\_s: 3\*\* - Critical mathematical structure

- \*\*S: 3\*\* - Mathematical types essential

- \*\*M: 3\*\* - Mature mathematical libraries

- \*\*D: 3\*\* - Deterministic mathematical computation

- \*\*E: 3\*\* - A priori mathematical proofs

- \*\*Hash: [1, 3, 3, 3, 3, 3]\*\*

\*\*14. Model checking (PyNuSMV)\*\*

- \*\*R\_c: 1\*\* - Low compression (state space exploration)

- \*\*R\_s: 3\*\* - Critical state machine structure

- \*\*S: 3\*\* - Formal logic types essential

- \*\*M: 3\*\* - Mature verification techniques

- \*\*D: 3\*\* - Deterministic verification

- \*\*E: 3\*\* - A priori formal verification theory

- \*\*Hash: [1, 3, 3, 3, 3, 3]\*\*

\*\*15. Experta (expert system)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through rule patterns

- \*\*R\_s: 2\*\* - Moderate structure (rule organization)

- \*\*S: 2\*\* - Rule-based reasoning dominates

- \*\*M: 3\*\* - Mature expert system techniques

- \*\*D: 2\*\* - Stable rule-based inference

- \*\*E: 2\*\* - Largely derivable from logic

- \*\*Hash: [2, 2, 2, 3, 2, 2]\*\*

\*\*16. Creative arts generative AI\*\*

- \*\*R\_c: 1\*\* - Low compression (creative diversity)

- \*\*R\_s: 0\*\* - Minimal structure preservation needed

- \*\*S: 0\*\* - Extensional/example-based generation

- \*\*M: 1\*\* - Heuristics exist but rapidly evolving

- \*\*D: 0\*\* - Highly non-deterministic creative process

- \*\*E: 0\*\* - A posteriori aesthetic/creative knowledge

- \*\*Hash: [1, 0, 0, 1, 0, 0]\*\*

[1 tool called]

### Learning Tools

\*\*17. Research with crawl4AI\*\*

- \*\*R\_c: 2\*\* - Moderate compression through content extraction

- \*\*R\_s: 1\*\* - Light structure (HTML/content patterns help)

- \*\*S: 1\*\* - Large enumerable web content

- \*\*M: 2\*\* - Standard web scraping with AI enhancement

- \*\*D: 1\*\* - Weak structure due to web variability

- \*\*E: 0\*\* - A posteriori web content requiring citations

- \*\*Hash: [2, 1, 1, 2, 1, 0]\*\*

\*\*18. General research tools (Perplexity, OpenAI Research, SearXNG, etc.)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through search/summarization

- \*\*R\_s: 1\*\* - Light structure (search result organization)

- \*\*S: 1\*\* - Large enumerable knowledge base

- \*\*M: 3\*\* - Mature search and information retrieval

- \*\*D: 1\*\* - Weak structure due to information diversity

- \*\*E: 0\*\* - A posteriori knowledge requiring citations

- \*\*Hash: [2, 1, 1, 3, 1, 0]\*\*

\*\*19. Tool for upskilling in a domain / creating new tools\*\*

- \*\*R\_c: 2\*\* - Moderate compression through learning patterns

- \*\*R\_s: 2\*\* - Moderate structure (curriculum, skill trees)

- \*\*S: 2\*\* - Learning rules and pedagogical constraints

- \*\*M: 1\*\* - Heuristics exist but personalized learning is evolving

- \*\*D: 1\*\* - Weak structure due to individual differences

- \*\*E: 1\*\* - Mixed (learning theory + empirical outcomes)

- \*\*Hash: [2, 2, 2, 1, 1, 1]\*\*

\*\*20. AutoML for perceptual/statistical tasks\*\*

- \*\*R\_c: 2\*\* - Moderate compression through model selection

- \*\*R\_s: 2\*\* - Moderate structure (pipeline organization)

- \*\*S: 2\*\* - ML rules and hyperparameter constraints

- \*\*M: 3\*\* - Mature AutoML libraries and frameworks

- \*\*D: 2\*\* - Stable after feature engineering transforms

- \*\*E: 2\*\* - Largely derivable from statistical theory

- \*\*Hash: [2, 2, 2, 3, 2, 2]\*\*

\*\*21. Knowledge/heuristic discovery (Eurisko), data mining, pattern identification via ML\*\*

- \*\*R\_c: 3\*\* - High compression through pattern discovery

- \*\*R\_s: 2\*\* - Moderate structure (pattern representations)

- \*\*S: 2\*\* - Pattern rules and constraints dominate

- \*\*M: 2\*\* - Standard ML algorithms with discovery heuristics

- \*\*D: 1\*\* - Weak structure due to discovery uncertainty

- \*\*E: 1\*\* - Mixed (some theory, much empirical)

- \*\*Hash: [3, 2, 2, 2, 1, 1]\*\*

\*\*22. Markitdown (markdown + OCR + formula/diagram understanding)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through structured parsing

- \*\*R\_s: 2\*\* - Moderate structure (document structure preservation)

- \*\*S: 1\*\* - Large enumerable document formats

- \*\*M: 2\*\* - Standard OCR/parsing with AI enhancement

- \*\*D: 2\*\* - Stable document processing patterns

- \*\*E: 1\*\* - Mixed (parsing theory + empirical recognition)

- \*\*Hash: [2, 2, 1, 2, 2, 1]\*\*

\*\*23. Langextract (structured text extraction)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through structured extraction

- \*\*R\_s: 3\*\* - Critical structure preservation (schemas, entities)

- \*\*S: 2\*\* - Extraction rules and patterns dominate

- \*\*M: 2\*\* - Standard NLP with structured output

- \*\*D: 2\*\* - Stable extraction patterns

- \*\*E: 1\*\* - Mixed (NLP theory + empirical patterns)

- \*\*Hash: [2, 3, 2, 2, 2, 1]\*\*

\*\*24. Antifragility MCP (feedback-based boosting)\*\*

- \*\*R\_c: 1\*\* - Low compression (feedback loops are complex)

- \*\*R\_s: 2\*\* - Moderate structure (feedback mechanisms)

- \*\*S: 2\*\* - Adaptation rules and feedback constraints

- \*\*M: 1\*\* - Heuristics exist but emerging field

- \*\*D: 1\*\* - Weak structure due to adaptive complexity

- \*\*E: 2\*\* - Largely derivable from systems theory

- \*\*Hash: [1, 2, 2, 1, 1, 2]\*\*

[1 tool called]

### Agentic/System Tools

\*\*25. Debugging/programming tools\*\*

- \*\*R\_c: 1\*\* - Low compression (debugging traces are verbose)

- \*\*R\_s: 2\*\* - Moderate structure (call stacks, breakpoints)

- \*\*S: 2\*\* - Debugging rules and program logic constraints

- \*\*M: 3\*\* - Mature debugging methodologies and tools

- \*\*D: 2\*\* - Stable debugging patterns

- \*\*E: 2\*\* - Largely derivable from program semantics

- \*\*Hash: [1, 2, 2, 3, 2, 2]\*\*

\*\*26. File manipulation/shell scripting\*\*

- \*\*R\_c: 2\*\* - Moderate compression through scripting patterns

- \*\*R\_s: 1\*\* - Light structure (file system hierarchies help)

- \*\*S: 1\*\* - Large enumerable file operations

- \*\*M: 3\*\* - Mature shell scripting and file operations

- \*\*D: 3\*\* - Highly deterministic file operations

- \*\*E: 3\*\* - A priori file system semantics

- \*\*Hash: [2, 1, 1, 3, 3, 3]\*\*

\*\*27. API/MCP connections\*\*

- \*\*R\_c: 2\*\* - Moderate compression through protocol standards

- \*\*R\_s: 3\*\* - Critical protocol structure preservation

- \*\*S: 2\*\* - API rules and protocol constraints dominate

- \*\*M: 3\*\* - Mature networking and API standards

- \*\*D: 3\*\* - Deterministic protocol behavior

- \*\*E: 3\*\* - A priori protocol specifications

- \*\*Hash: [2, 3, 2, 3, 3, 3]\*\*

\*\*28. Start/stop/handoff/guardrails/resource constraints; call agents (Autogen, Agents SDK, PydanticAI)\*\*

- \*\*R\_c: 1\*\* - Low compression (orchestration is complex)

- \*\*R\_s: 3\*\* - Critical orchestration structure

- \*\*S: 2\*\* - Orchestration rules and constraints dominate

- \*\*M: 2\*\* - Standard patterns emerging in agent orchestration

- \*\*D: 2\*\* - Stable orchestration patterns

- \*\*E: 2\*\* - Largely derivable from distributed systems theory

- \*\*Hash: [1, 3, 2, 2, 2, 2]\*\*

\*\*29. RQ (workers)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through job queuing

- \*\*R\_s: 2\*\* - Moderate structure (queue organization)

- \*\*S: 1\*\* - Large enumerable job types

- \*\*M: 3\*\* - Mature job queue systems

- \*\*D: 3\*\* - Deterministic job processing

- \*\*E: 3\*\* - A priori queuing theory

- \*\*Hash: [2, 2, 1, 3, 3, 3]\*\*

\*\*30. Execute code in sandbox\*\*

- \*\*R\_c: 1\*\* - Low compression (execution context is complex)

- \*\*R\_s: 2\*\* - Moderate structure (sandbox boundaries)

- \*\*S: 2\*\* - Execution rules and security constraints

- \*\*M: 3\*\* - Mature containerization and sandboxing

- \*\*D: 2\*\* - Stable execution patterns with some variability

- \*\*E: 2\*\* - Largely derivable from OS and security theory

- \*\*Hash: [1, 2, 2, 3, 2, 2]\*\*

\*\*31. Computer-use agent (Playwright MCP, OpenCUA)\*\*

- \*\*R\_c: 1\*\* - Low compression (UI interactions are complex)

- \*\*R\_s: 1\*\* - Light structure (UI patterns help but vary)

- \*\*S: 1\*\* - Large enumerable UI interactions

- \*\*M: 2\*\* - Standard automation tools with AI enhancement

- \*\*D: 1\*\* - Weak structure due to UI variability

- \*\*E: 1\*\* - Mixed (automation theory + empirical UI patterns)

- \*\*Hash: [1, 1, 1, 2, 1, 1]\*\*

\*\*32. Activepieces (business automation)\*\*

- \*\*R\_c: 2\*\* - Moderate compression through workflow patterns

- \*\*R\_s: 2\*\* - Moderate structure (workflow organization)

- \*\*S: 2\*\* - Business rules and workflow constraints

- \*\*M: 3\*\* - Mature workflow automation platforms

- \*\*D: 2\*\* - Stable workflow patterns

- \*\*E: 2\*\* - Largely derivable from business process theory

- \*\*Hash: [2, 2, 2, 3, 2, 2]\*\*

\*\*33. LiteLLM for evaluating system performance/telemetry\*\*

- \*\*R\_c: 2\*\* - Moderate compression through metrics aggregation

- \*\*R\_s: 2\*\* - Moderate structure (metrics organization)

- \*\*S: 1\*\* - Large enumerable performance metrics

- \*\*M: 3\*\* - Mature monitoring and telemetry systems

- \*\*D: 3\*\* - Deterministic metrics collection

- \*\*E: 2\*\* - Largely derivable from systems theory

- \*\*Hash: [2, 2, 1, 3, 3, 2]\*\*

\*\*34. Homoiconic tool (tool for acting on/modifying self)\*\*

- \*\*R\_c: 1\*\* - Low compression (self-modification is complex)

- \*\*R\_s: 3\*\* - Critical meta-structure preservation

- \*\*S: 3\*\* - Meta-types and self-reference essential

- \*\*M: 1\*\* - Heuristics exist but highly experimental

- \*\*D: 1\*\* - Weak structure due to self-modification complexity

- \*\*E: 2\*\* - Largely derivable from formal meta-theory

- \*\*Hash: [1, 3, 3, 1, 1, 2]\*\*

[1 tool called]

## Complete Tool Hash Matrix

Here's the complete analysis with 5-axis hashes for all tools, organized by category:

### Memory Tools

| Tool | R\_c | R\_s | S | M | D | E | Hash |

|------|-----|-----|---|---|---|---|------|

| Docs indexing/Vector DB | 3 | 2 | 1 | 3 | 2 | 0 | [3,2,1,3,2,0] |

| Memory MDP (Memento) | 1 | 3 | 2 | 2 | 2 | 2 | [1,3,2,2,2,2] |

| PostgreSQL Database | 2 | 3 | 2 | 3 | 3 | 2 | [2,3,2,3,3,2] |

| Regex/Grep | 3 | 1 | 2 | 3 | 3 | 3 | [3,1,2,3,3,3] |

| Dependency analysis | 2 | 3 | 2 | 3 | 3 | 3 | [2,3,2,3,3,3] |

| Text compression | 3 | 1 | 1 | 3 | 3 | 3 | [3,1,1,3,3,3] |

### Reasoning Tools

| Tool | R\_c | R\_s | S | M | D | E | Hash |

|------|-----|-----|---|---|---|---|------|

| Taskmaster | 2 | 3 | 2 | 2 | 2 | 1 | [2,3,2,2,2,1] |

| Smolagents (thought-as-code) | 1 | 2 | 3 | 1 | 1 | 2 | [1,2,3,1,1,2] |

| ROMA (hierarchical planning) | 2 | 3 | 2 | 2 | 2 | 1 | [2,3,2,2,2,1] |

| DSPy + Pydantic | 1 | 3 | 3 | 2 | 3 | 3 | [1,3,3,2,3,3] |

| Spec kit | 2 | 3 | 3 | 2 | 3 | 3 | [2,3,3,2,3,3] |

| Category theory tools | 1 | 3 | 3 | 2 | 3 | 3 | [1,3,3,2,3,3] |

| Math tools (SymPy/Z3/OR-Tools) | 1 | 3 | 3 | 3 | 3 | 3 | [1,3,3,3,3,3] |

| Model checking (PyNuSMV) | 1 | 3 | 3 | 3 | 3 | 3 | [1,3,3,3,3,3] |

| Experta (expert system) | 2 | 2 | 2 | 3 | 2 | 2 | [2,2,2,3,2,2] |

| Creative arts generative AI | 1 | 0 | 0 | 1 | 0 | 0 | [1,0,0,1,0,0] |

### Learning Tools

| Tool | R\_c | R\_s | S | M | D | E | Hash |

|------|-----|-----|---|---|---|---|------|

| Crawl4AI research | 2 | 1 | 1 | 2 | 1 | 0 | [2,1,1,2,1,0] |

| General research tools | 2 | 1 | 1 | 3 | 1 | 0 | [2,1,1,3,1,0] |

| Upskilling/tool creation | 2 | 2 | 2 | 1 | 1 | 1 | [2,2,2,1,1,1] |

| AutoML | 2 | 2 | 2 | 3 | 2 | 2 | [2,2,2,3,2,2] |

| Eurisko/pattern discovery | 3 | 2 | 2 | 2 | 1 | 1 | [3,2,2,2,1,1] |

| Markitdown | 2 | 2 | 1 | 2 | 2 | 1 | [2,2,1,2,2,1] |

| Langextract | 2 | 3 | 2 | 2 | 2 | 1 | [2,3,2,2,2,1] |

| Antifragility MCP | 1 | 2 | 2 | 1 | 1 | 2 | [1,2,2,1,1,2] |

### Agentic/System Tools

| Tool | R\_c | R\_s | S | M | D | E | Hash |

|------|-----|-----|---|---|---|---|------|

| Debugging/programming | 1 | 2 | 2 | 3 | 2 | 2 | [1,2,2,3,2,2] |

| File manipulation/shell | 2 | 1 | 1 | 3 | 3 | 3 | [2,1,1,3,3,3] |

| API/MCP connections | 2 | 3 | 2 | 3 | 3 | 3 | [2,3,2,3,3,3] |

| Agent orchestration | 1 | 3 | 2 | 2 | 2 | 2 | [1,3,2,2,2,2] |

| RQ workers | 2 | 2 | 1 | 3 | 3 | 3 | [2,2,1,3,3,3] |

| Sandbox execution | 1 | 2 | 2 | 3 | 2 | 2 | [1,2,2,3,2,2] |

| Computer-use agent | 1 | 1 | 1 | 2 | 1 | 1 | [1,1,1,2,1,1] |

| Activepieces | 2 | 2 | 2 | 3 | 2 | 2 | [2,2,2,3,2,2] |

| LiteLLM telemetry | 2 | 2 | 1 | 3 | 3 | 2 | [2,2,1,3,3,2] |

| Homoiconic tool | 1 | 3 | 3 | 1 | 1 | 2 | [1,3,3,1,1,2] |

## Usage for Tool Selection

To select the best tool for a task:

1. \*\*Score your task\*\* using the same 5-axis framework

2. \*\*Calculate distance\*\* using L0, L1, or L2 norm:

- \*\*L0 norm\*\*: Count of non-zero differences (discrete matching)

- \*\*L1 norm\*\*: Sum of absolute differences (Manhattan distance)

- \*\*L2 norm\*\*: Square root of sum of squared differences (Euclidean distance)

3. \*\*Select the tool\*\* with minimum distance to your task

\*\*Example\*\*: For a task scored as [2,2,2,2,2,2], you would calculate distances to all tool hashes and select the closest match.

This systematic approach allows you to objectively match tools to tasks based on their fundamental algorithmic characteristics rather than superficial descriptions.

—-------------------------------------------------

### Critique of the 5-axis algorithm (with code references)

- The rubrics are crisp and useful, but several axes are interdependent in practice while the algorithm treats them as independent.

- R\_c and R\_s can correlate: highly structured data often compresses better when encoded; treating them independently can mis-rank tasks that are both compressible and structure-critical.

- S interacts with E and D: intensional/type-heavy tasks (S=3) tend to raise E and D due to stronger formal guarantees.

- Boundaries are coarse, which is great for robustness but can be blunt for nuanced choices. Consider soft thresholds or calibrated sub-bins.

- The “rationale” fields aren’t used downstream; capturing them as features could improve selection when distances tie.

- The final planner rightly uses constraints and budgets to transform steps, but tool selection based solely on nearest-hash may underweight budgets and risk. A blended scoring with budget/risk should steer tool choice.

Code references for rubric anchors:

```273:281:web\_app\_reference/strategy\_planner\_hardened.py

"Assess representational reducibility and structure-fidelity.

Definitions:

- Compression ratio (CR): length(original) / length(optimal lossless compression).

- Structure-fidelity need (R\_s): degree to which explicit structure (schema/AST/graph) is required.

Rubric:

- R\_c: 3 if CR >= 2.00; 2 if 1.40 <= CR < 2.00; 1 if 1.10 <= CR < 1.40; 0 if CR < 1.10.

- R\_s: 3 if preserve >= 80% key fields/relations and >= 3 critical invariants; 2 if 50-79% and 1-2 invariants; 1 if ad-hoc structure helps but not required; 0 if none.

```

```319:322:web\_app\_reference/strategy\_planner\_hardened.py

Rubric S: 0 extensional (finite/exhaustive by examples), 1 large-but-enumerable, 2 rules/constraints dominate, 3 intensional (types essential).

```

```362:363:web\_app\_reference/strategy\_planner\_hardened.py

Assess problem maturity (M). 0 open; 1 heuristics exist; 2 standard algorithms with partial proofs/benchmarks; 3 solved with strong libraries and consensus.

```

```398:400:web\_app\_reference/strategy\_planner\_hardened.py

Assess determinism/modelability (D). 0 irreducible/non-stationary; 1 weak structure; 2 stable after transforms; 3 reducible with parsimonious model and R^2 >= 0.80.

```

```440:441:web\_app\_reference/strategy\_planner\_hardened.py

Assess epistemic grounding (E). 0 a posteriori (must cite); 1 mixed; 2 largely derivable; 3 a priori (proofs/model checking feasible).

```

### Critique of specific tool assignments

- Regex/Grep [3,1,2,3,3,3]: R\_s might be 0 for raw grep (no structure preservation), but 1 is reasonable if patterns encode minimal structure. Consider splitting “regex” (R\_s=1) vs “grep” (R\_s=0).

- Creative arts generative AI [1,0,0,1,0,0]: R\_c likely 0—outputs are entropy-rich and not compressible in the lossless sense. Adjust to [0,0,0,1,0,0].

- Taskmaster [2,3,2,2,2,1]: E might be 2 in strictly spec-driven use (derivable plans), but 1 when research loops are used. Suggest two profiles: “strict-spec” [2,3,2,2,2,2] vs “research-heavy” [2,3,2,2,2,1].

- Smolagents [1,2,3,1,1,2]: Some deployments enforce tight unit-test loops, improving D from 1→2. Consider environment-specific profile.

- Vector DB [3,2,1,3,2,0]: When used with metadata schemas and chunk-link invariants, R\_s can be 3. Consider “schema-aware RAG”: [3,3,1,3,2,0].

- Dependency analysis [2,3,2,3,3,3]: If probabilistic static analysis is used (e.g., inferred edges), D could dip to 2.

### How it performs on typical agentic tasks

- Task A: “Summarize a 200-page PDF and answer questions with citations.”

- Task vector: R\_c=3 (compressible), R\_s=2 (doc structure helps), S=1, M=3, D=2, E=0 → [3,2,1,3,2,0]

- Nearest tools (L2/L1): Vector DB [3,2,1,3,2,0] (exact match), Markitdown [2,2,1,2,2,1], Langextract [2,3,2,2,2,1]

- Outcome: Good; pipeline of Markitdown → Langextract → Vector DB hits both structure and compressibility. Using only nearest (Vector DB) risks losing layout math; multi-tool blending is better.

- Task B: “Refactor a Python monolith into packages while preserving import invariants.”

- Task vector: R\_c=1, R\_s=3 (invariants), S=2, M=3, D=3, E=3 → [1,3,2,3,3,3]

- Nearest tools: Dependency analysis [2,3,2,3,3,3], PostgreSQL (for cataloging) [2,3,2,3,3,2], Spec kit [2,3,3,2,3,3]

- Outcome: Strong; dependency analysis should win. Spec kit helps formalize invariants. Consider pushing E up via model checking when invariants are safety-critical.

- Task C: “Design and verify a scheduling constraint for job assignment.”

- Task vector: R\_c=1, R\_s=3, S=3, M=3, D=3, E=3 → [1,3,3,3,3,3]

- Nearest tools: OR-Tools/SymPy/Z3 [1,3,3,3,3,3], Model checking [1,3,3,3,3,3], Spec kit [2,3,3,2,3,3]

- Outcome: Excellent; exact matches. Spec kit as adjunct to formal solver is appropriate.

- Task D: “Plan a multi-week feature (payments) with risks and dependencies.”

- Task vector: R\_c=2, R\_s=3, S=2, M=2, D=2, E=1 → [2,3,2,2,2,1]

- Nearest tools: Taskmaster [2,3,2,2,2,1], ROMA [2,3,2,2,2,1], PostgreSQL [2,3,2,3,3,2]

- Outcome: Good; Taskmaster/ROMA preferred. Postgres is supportive for artifact/state tracking but not the planner.

- Task E: “Generate creative marketing copy across 10 variants.”

- Task vector: R\_c=0, R\_s=0, S=0, M=1, D=0, E=0 → [0,0,0,1,0,0]

- Nearest tools: Creative AI [1,0,0,1,0,0] or adjusted [0,0,0,1,0,0]

- Outcome: Works if R\_c is corrected to 0.

- Task F: “Extract entities from invoices and store in a relational DB.”

- Task vector: R\_c=2, R\_s=3, S=2, M=2, D=2, E=1 → [2,3,2,2,2,1]

- Nearest tools: Langextract [2,3,2,2,2,1], PostgreSQL [2,3,2,3,3,2], Markitdown [2,2,1,2,2,1]

- Outcome: Solid pipeline: Markitdown → Langextract → PostgreSQL. If strict schemas, consider Spec kit to raise E.

### Distance metric behavior (why it matters)

- L0 favors exact category matches; brittle for near-misses (e.g., Langextract vs Vector DB differs in R\_s and S by 1, but L0=2 equals others).

- L1 is robust for discrete scales (0–3) and interpretable—use as default.

- L2 over-penalizes a few large mismatches; useful when single-axis errors are critical (e.g., E misalignment for safety tasks).

### Recommendations

- Add weights per environment: w\_E high for regulated domains; w\_R\_s high for data pipelines; w\_D high for ops/reliability.

- Maintain multiple profiles for tools used in different modes (e.g., Vector DB “schema-aware” vs “pure”; Taskmaster “spec” vs “research”).

- Prefer top-2 blending when the nearest tool doesn’t cover both R\_s and E (e.g., Vector DB + Langextract).

- Calibrate hashes with telemetry: log chosen tool vs task vector and success metrics to refine scores.

- Incorporate “rationale” strings into tie-breaking: prefer the tool whose rationale tokens best match task invariants/constraints.

- Allow budget/risk overlays to veto a nearest tool if it violates time/memory constraints or risk thresholds (aligns with plan\_budgeted and risk\_aware\_plan).

- Minor hash tweaks to apply now:

- Creative AI → [0,0,0,1,0,0]

- Split Regex vs Grep (Regex: [3,1,2,3,3,3], Grep: [3,0,1,3,3,3])

- Taskmaster: add “spec-mode” secondary profile [2,3,2,2,2,2]

- Vector DB: add “schema-aware” secondary profile [3,3,1,3,2,0]

- Practical default:

- Use L1 with per-domain weights, select top-2 tools, then check budget/risk overlays before finalizing.

—----------------------------C4.5 Decision Tree

### C4.5-style decision tree for tool selection (LLM-askable)

Answer each question; follow the first matching path to a leaf. At leaves, select the first tool; include the next as backup. If multiple apply, return the top-2 and compose a pipeline.

- Q0. Is this primarily an execution/ops task (debugging, scripting, API wiring, orchestration, workers, sandbox, browser automation, telemetry)?

- Yes →

- If debugging/program analysis → Debugging/programming tools

- If file/data plumbing → File manipulation/shell scripting

- If protocol/tool wiring → API/MCP connections

- If multi-agent runtime control → Agent orchestration (Autogen/Agents SDK/PydanticAI)

- If background jobs → RQ (workers)

- If safe execution required → Execute code in sandbox

- If desktop/web UI control → Computer-use agent (Playwright MCP/OpenCUA)

- If business workflows → Activepieces

- If eval/metrics → LiteLLM telemetry

- No → continue

- Q1. Do you require formal guarantees or a priori correctness (proofs/model checking feasible)? E ≥ 2

- Yes →

- Q1.1. Is the task constraint/optimization/scheduling/SMT? S ≥ 2

- Yes →

- If optimization/scheduling → Math tools (OR-Tools/Z3)

- If temporal/state properties → Model checking (PyNuSMV)

- Always add Spec kit for formal specs; optionally dspy+pydantic for typed scaffolding

- No →

- If symbolic algebra/calculus/derivation → Math tools (SymPy/SageMath)

- If abstract structural modeling → Category theory tools (Catlab.jl) + Spec kit

- No → continue

- Q2. Is external evidence/citations required (answers must cite sources)? E = 0

- Yes →

- Q2.1. Is the goal creative generation (copy/imagery/music) without strict structure? S = 0 and D = 0

- Yes → Creative arts generative AI

- No → Research stack:

- If web needed → General research tools (Perplexity/SearXNG/OpenAI Research) ± crawl4AI

- If persistent retrieval helpful → Docs indexing/Vector DB (schema-aware if available)

- No → continue

- Q3. How much structure must be preserved? R\_s

- High (R\_s ≥ 2) →

- Q3.1. Is the modality documents (PDF/Markdown/images/math/diagrams)?

- Yes → Markitdown → Langextract → PostgreSQL (schema) → (optional) Vector DB

- Q3.2. Is it codebase/package/graph relations?

- Yes → Dependency analysis → (optional) Spec kit for invariants

- Q3.3. Is it long-term memory (episodic/procedural/semantic)?

- Yes → PostgreSQL (relational schemas for memory) ± Vector DB for semantic recall

- Low (R\_s ≤ 1) →

- Q3.4. Is the content highly compressible or patternable? R\_c ≥ 2

- Yes → Regex (structured patterns) or Grep (raw search) → (optional) Advanced text compression → Vector DB for fuzzy recall

- No → continue

- Q4. What semantic mode best fits? S

- Extensional / large-but-enumerable (S ∈ {0,1}) →

- If large corpus recall → Docs indexing/Vector DB

- If lookup/nearest-neighbor fits → KNN/lookup (Vector DB) ± Taskmaster enumeration

- Intensional / type- or rule-driven (S ∈ {2,3}) →

- Q4.1. Is the domain mature? M ≥ 2

- Yes → Spec kit + dspy+pydantic (typed logic) → Math/logic tools as needed

- No → ROMA (hierarchical planning) or Taskmaster with research loops

- Q5. How predictable is the process? D

- High/stable after transforms (D ≥ 2) →

- Prefer model-based approaches: AutoML (for perception/stats), OR-Tools (constraints), PostgreSQL constraints, derive\_model\_with\_bounds

- Low/weak structure (D ≤ 1) →

- Prefer explore/simulate/adapt: Thought-as-code (smolagents), Antifragility MCP (feedback boosting), simulate/generate-and-test loops

- Q6. Maturity override (final check). M

- Mature (M ≥ 2) → Prefer established libs: PostgreSQL, Regex/grep, Dependency analysis, OR-Tools/Z3, Model checking

- Emerging (M ≤ 1) → Prefer learning/research: AutoML, Eurisko/pattern discovery, Research tools

Tie-breakers and overlays

- If multiple leaves apply, compute L1 distance between task hash [R\_c,R\_s,S,M,D,E] and tool hashes; pick top-2; compose.

- Apply budget/risk overlays: if over budget, downshift to cheaper tool; if risk high, add proofs/typecheck/uncertainty steps (Spec kit, model checking, estimate\_uncertainty).

- If R\_s high and E low, prefer extraction→DB before retrieval; if E low and S low, prefer retrieval→generation.

Example leaves (pipelines)

- “Summarize and cite a long PDF”: Markitdown → Langextract → Vector DB → Research tool

- “Refactor monolith safely”: Dependency analysis → Spec kit → Model checking (if safety-critical)

- “Build scheduling constraint”: Spec kit → OR-Tools/Z3 → (optional) Model checking

- “Cold-start document QA without schemas”: Vector DB → (optional) Regex patterns → Research tool

This tree is designed for an LLM to traverse using the task’s axis scores (or by answering the questions directly), returning a concrete tool or 2-tool pipeline aligned to structure, epistemic needs, maturity, determinism, and compressibility.

—-----------------------------------------

### 10 tasks, tool choices via the tree, and what we learned

1) Legal doc Q&A with citations (long PDF)

- Task hash: [3,2,1,3,2,0]

- Picked: Markitdown → Langextract → Vector DB → Research tool

- Notes: Strong match. Pitfall: if tables/math heavy, need “schema-aware RAG” profile or extraction recall degrades.

2) E‑commerce recommender (nearest-neighbor recall + metadata)

- Task hash: [3,2,1,3,2,0]

- Picked: Vector DB (schema-aware) → Taskmaster (enumeration)

- Notes: Works; tree underweights business constraints (E≈1) and exploration budget—should surface AutoML as optional branch.

3) Invoice ETL to relational DB

- Task hash: [2,3,2,2,2,1]

- Picked: Markitdown → Langextract → PostgreSQL (+ Spec kit if strict)

- Notes: Good. Failure mode: low-quality scans; need fallback to human-in-loop or OCR confidence gating.

4) Monolith refactor preserving import invariants

- Task hash: [1,3,2,3,3,3]

- Picked: Dependency analysis → Spec kit (invariants) → (optional) Model checking

- Notes: Accurate. If dynamic imports, D may dip to 2; tree could adapt by proposing tracing.

5) Job-shop scheduling with SLAs

- Task hash: [1,3,3,3,3,3]

- Picked: Spec kit → OR-Tools/Z3 → (optional) Model checking

- Notes: Exact match. Budget overlays matter; large instances may need heuristic hybrid.

6) Creative marketing copy with variants

- Task hash: [0,0,0,1,0,0]

- Picked: Creative arts generative AI

- Notes: Correct. If brand/legal constraints rise (E→1-2), add retrieval guardrails or policy-checker.

7) Production incident debugging (logs + hypothesis)

- Task hash: [1,2,2,3,2,2]

- Picked: Debugging tools → Regex for log slicing → LiteLLM telemetry

- Notes: Good start; if weak structure (D→1), tree should pivot to Antifragility MCP or simulate/generate‑and‑test.

8) FAQ chatbot over internal wiki

- Task hash: [3,2,1,3,2,0]

- Picked: Vector DB → (optional) Langextract → Taskmaster for content ops

- Notes: Works. If access control/PII constraints (E→2), add Spec kit for policy checks.

9) Literature review synthesis with citations

- Task hash: [2,1,1,3,1,0]

- Picked: Research tools → crawl4AI → Vector DB (scratchpad)

- Notes: Solid. Tree should prioritize dedup/quality filters; add confidence/ranking step.

10) Autonomous UI form filling (RPA)

- Task hash: [1,1,1,2,1,1]

- Picked: Computer-use agent → File/Shell for artifacts → API/MCP if available

- Notes: Fragile to UI drift (D=1). Tree should suggest visual anchors and regression snapshots.

### Critical evaluation: issues surfaced

- Under-specified multi-profile tools: Vector DB (pure vs schema-aware), Taskmaster (spec vs research), Regex vs Grep. Add explicit profiles.

- Budget/risk overlays must run before final choice: some picks are correct but impractical at scale (e.g., model checking on large systems, OR-Tools on huge jobs).

- E/S coupling: when E is low but S is high (e.g., ETL), tree should prefer extraction→DB before retrieval; current path does this implicitly but not forcefully.

- D sensitivity: volatile domains (UI, incidents) need adaptive/feedback branches (Antifragility MCP, simulate/generate‑and‑test) promoted earlier.

- Creative tasks: R\_c should be 0; corrected path works better. The tree must avoid compressibility heuristics for generative tasks.

- OCR/noisy inputs: add confidence gates and human-in-loop fallback explicitly.

- Domain weighting: regulated domains need higher E weight; ops need higher D weight. Hard-coded branch order isn’t enough.

### Recommended refinements

- Add tool profiles:

- Vector DB: {pure, schema-aware}

- Taskmaster: {spec, research-heavy}

- Regex vs Grep split

- Insert overlays before commitment:

- Budget: downshift to heuristics/hybrids; shrink samples; approximate solvers

- Risk: add Spec kit/typecheck/estimate\_uncertainty/model checking

- Confidence (OCR/extraction): route to review loop when low

- Promote D-aware branches earlier:

- If D≤1, surface Antifragility MCP and smolagents explore-then-commit

- Telemetry calibration:

- Log task hash → selected tools → outcome; periodically adjust hashes/weights

- Tie-breaking:

- Default L1 distance with domain weights; top‑2 tool composition; use rationale text for semantic tie-breaks

- Net: The tree works well for structured/data/ops/formal tasks, needs explicit profiles, budget/risk/confidence overlays, and stronger D/E-aware routing for volatile/creative/regulated scenarios.

—-----------------

Basically the recipe is get the relevant information into a RAG pipeline and then use a workflow + react to produce a solution. Eval it. Iterate and repeat over a hierarchy of subgoals for some larger goal with dependency and context management and learning. You then have a MIVU.