Wonton Soup: Proof Structures Under Interventions

Specter Labs

1. Wonton Soup: Proof Structures Under Interventions

`wonton-soup` is our intervention harness for proof-search experiments. The question: when we perturb a solver's search process, does it return to the same proof structure, or settle into a different one?

We run wild-type and intervention sweeps over deterministic theorem samples, capture full search artifacts (`*_history.json`, `*_mcts_tree.json`, `*_graph.json`, `*_comparison.json`), and compare outcomes across seeds, tactics, providers, and backends.

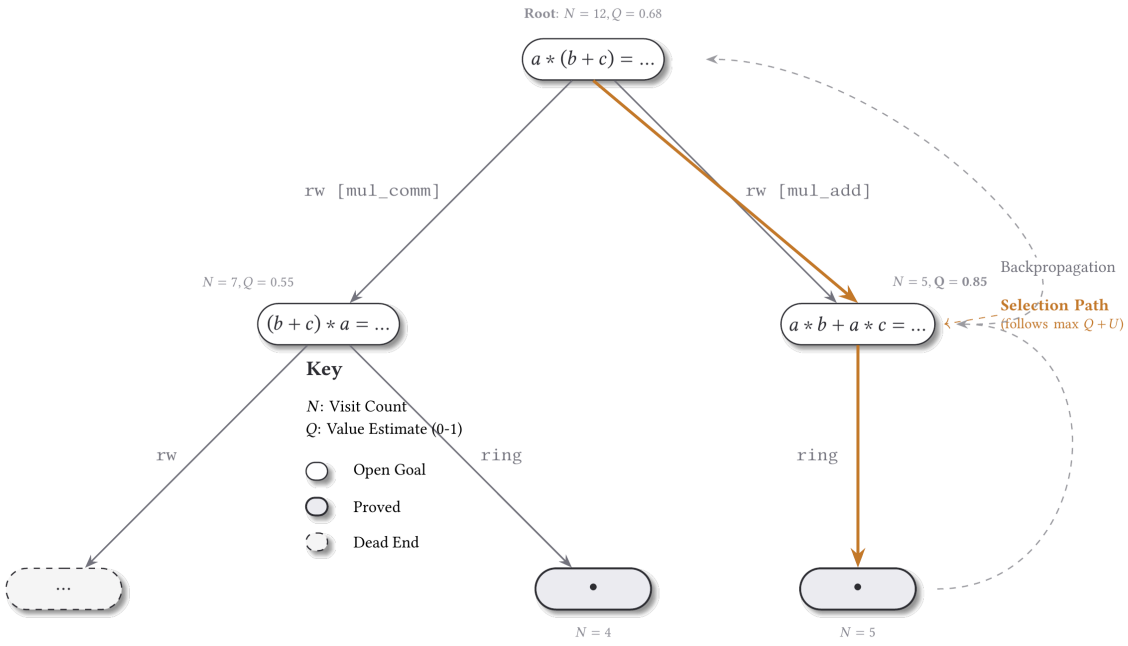

Figure 1: MCTS Proof Search Tree

1.1. 1. What We're Looking For

We treat proof search as a stochastic process over structured states.

- A perturbation can be a blocked tactic or tactic family, a seed change, or a policy/scheduler change.
- A response can be recovery to the same recurring proof shape or migration into a different proof-graph cluster, which the basin analysis treats as an attractor.
- The object of study is not only solve rate; it is the shape and stability of search trajectories.

This is why we log enough structure to replay and compare runs months later under fixed configuration.

1.2. 2. Theoretical Mapping

Our intervention protocol applies three specific concepts from the Diverse Intelligence research program to formal proof search.

1.2.1. Search efficiency as a metric (K)

Following Chis-Ciure and Levin (2025), we treat intelligence as search efficiency in problem spaces, measuring the log-ratio between a random walk (τ_{blind}) and our observed agent (τ_{agent}):

$$K = \log_{10}(\tau_{blind}/\tau_{agent})$$

A positive K quantifies how many orders of magnitude our policy saves over a brute-force baseline.

1.2.2. Lesions and Rerouting

Zhang et al. (2024) demonstrate that decentralized systems, such as self-sorting arrays, can navigate around "damaged" components to reach a global goal, whereas our proof-search version blocks specific tactics or lemma families from a known solution path and measures whether the solver finds a different proof route or fails under the block.

1.2.3. Pattern Invariance (TAME)

The TAME framework (Levin, 2022) argues that behavioral structures, not just low-level mechanisms, persist under perturbation, and in `wonton-soup` we use Graph Edit Distance (GED) and basin analysis to determine whether proof search repeatedly settles into the same proof-graph clusters across different seeds and interventions.

1.3. 3. Harness and Corpus Design

Corpus and run configuration are saved as explicit inputs to every comparison: corpus builds are manifest-backed, run configs are snapshot-pinned, and downstream analysis reads those inputs directly, keeping the baseline fixed while only the intervention changes.

We also separate validity from capability: a theorem can be well-formed for a backend even if the provider cannot solve it under budget, so Gate A checks that an item is structurally processable

for the chosen backend and schema, whereas Gate B checks whether the provider and search policy can do meaningful work on that valid slice.

Deterministic selection (`--sample` with `--seed`) is how we ensure replayability, since rerunning later with the same corpus ref and selector inputs should recover the same theorem slice and comparable outputs.

Run-level schemas complete the provenance record with `run_config.json`, `run_status.json`, and `summary.json.gz` for postprocess, lake extraction, and cross-run audits.

1.3.1. What is fixed per comparison run

- Corpus reference plus build provenance (`manifest.json`, item ordering, hash identity).
- Selection procedure (`--sample`, `--seed`, `--offset`, `--limit`) and resulting theorem slice.
- Search budget and core execution knobs (mode, iteration budget, intervention declaration).
- Analysis inputs consumed by downstream tools (`run_config.json`, `run_status.json`, `summary.json.gz`, theorem subfiles).

1.4. 4. Search Core: Centralized and Distributed MCTS

Both modes walk a tactic-conditioned state graph and use compatible log formats, so a mode switch changes execution dynamics without changing what downstream analysis reads.

Centralized mode is the structural baseline, where a single global selection loop owns frontier choice and expansion order, gives one policy view over one queue, and minimizes coordination effects, making it the simplest setup for studying proof families, basin structure, reroute versus collapse, recovery after intervention, and blind-relative efficiency.

Distributed mode adds multiple workers over that same proof-search space, where local agents operate over a shared frontier, inflight reservations reduce duplicate expansion pressure, and scheduling controls let us change coordination directly: block, delay, reroute, virtual loss, and depth/path bias interventions change who explores what and when.

Centralized MCTS maps the proof-search space for a theorem slice, whereas distributed MCTS changes how workers choose and reserve parts of that same shared frontier, testing whether solve behavior depends on one expansion regime or stays robust when worker scheduling changes.

Both modes emit compatible tree and trace files, so comparisons can use the same analysis path (`*_mcts_tree.json`, traces, run summaries) instead of requiring mode-specific postprocess logic.

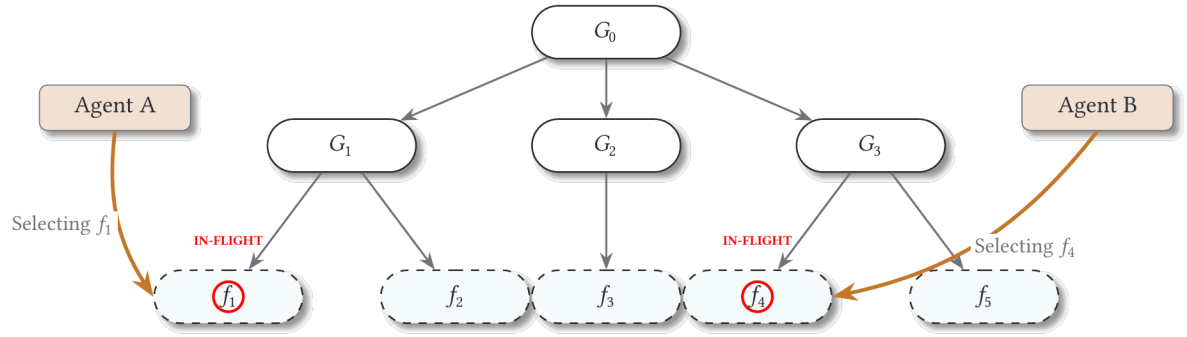

Figure 2: Distributed Frontier

How to read the figure: each worker lane represents local agent activity against a shared frontier, with reservations and scheduler policy shaping contention and handoff, while dense synchronized bands suggest strong coupling and staggered bands indicate looser parallel exploration.

1.5. 5. Backend Families and Artifact Compatibility

We use a multi-backend harness to test whether behavioral patterns persist across backends or are implementation artifacts. A pattern that recurs across backend families with different proof objects and trace outputs is a stronger candidate invariant.

`wonton-soup` currently supports five execution backends:

- `lean`
- `coq`
- `e`
- `vampire`
- `z3`

Run-level schemas are shared (`run_config.json`, `run_status.json`, `summary.json.gz`), and `run_status.json` flags plus file-presence checks say which outputs each backend can actually produce, so downstream analysis does not silently compare missing or incompatible files.

This matters for mixed analyses: `ged_search_graph` is meaningful only when a true search graph exists, whereas external solver traces may map to `ged_trace_graph` or proof-object comparisons instead, and capability flags plus validity metadata keep those distinctions visible.

1.5.1. Backend Output Types (Typical)

Backend	Search-graph output	Proof output	Trace output	Practical note

lean	ged_search_graph	proof-term artifacts when enabled	MCTS traces	Full search-graph comparisons are strongest here.
coq	usually unavailable	proof object family (integration dependent)	backend trace varies	Treat proof/trace availability as capability-gated.
e	unavailable	proof object family	ged_trace_graph from TSTP-style traces	Mark trace completeness explicitly.
vampire	unavailable	proof object family	optional trace family	Proof-centric comparison is typical.
z3	unavailable	proof object family	optional trace family	Search-graph GED is not the primary comparison.

Cross-backend comparisons are safest on shared run-level outcomes and explicitly labeled measurement types. Structure-level comparisons should be grouped by compatible output types, not collapsed into one undifferentiated score.

1.5.2. Showcase

We pin two recurring analysis views: attractor separation and blind-relative efficiency.
Attractor view: structural families and basin mass concentration. K view: efficiency over blind baseline for intervention comparisons.

1.6. 6. Metrics and Comparison Families

Each metric answers a different comparison question:

- K-style search efficiency (`k_search_efficiency`) from trace-derived blind nulls.
- Paper-style paired blind baseline (`paper_k`) from basin runs with `--basin-blind`.
- GED measurement types (`ged_search_graph`, `ged_search_graph_soft`, `ged_proof_graph`, `ged_trace_graph`) with explicit validity metadata.
- Trajectory comparison (divergence, reconvergence, recovery iterations).
- Basin analysis (solve rate, structure hash diversity, dominant basin frequency).
- Sheaf analyses (equivalence consistency and tactic-transform residuals).
- Cross-run lake exports for reproducible, cross-experiment aggregation.

1.6.1. Quick Metric Interpretation

Metric	What changed in the intervention run	How to read it
`k_search_efficiency` / `paper_k`	Attempted edge count before first solve (τ_{agent}) vs blind baseline (τ_{blind})	Higher is better; $K > 0$ means fewer attempts than blind
`normalized GED_search`	Search-graph structure relative to wild-type	Near `0` means structurally similar search; larger values mean stronger reroute
shared prefix	Number of early wild-type steps replayed before divergence	High prefix means late divergence; low prefix means early policy/path change
divergence iteration/depth	First step where intervention path differs	Lower means early structural perturbation; higher means late perturbation
solve status under block	Whether constrained run still reaches terminal proof	Distinguishes robust reroute from true tactic dependency
basin mass + attractor ID	Fraction of seeds ending in each clustered trajectory family	Concentrated mass indicates stable basin; split mass indicates multimodal behavior

K is reported as:

$$K = \log_{10}\left(\frac{\tau_{blind}}{\tau_{agent}}\right)$$

Example calibration: $K = \log_{10}(120/9) = 1.12$ (about $13\times$ fewer attempts than blind).

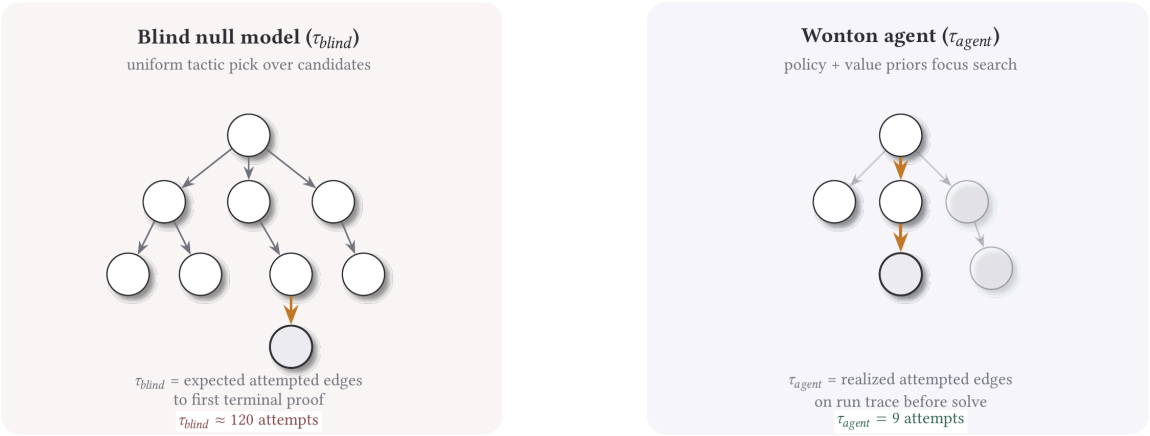

Figure 3: K-Metric Visualization

- τ_{agent}: attempted tactic edges until first terminal solve in the observed search graph.
- τ_{blind}: expected attempted edges for a matched blind null policy over the same available tactic choices.
- K: orders-of-magnitude efficiency over blind (`K > 0` is better than blind).

Two related outputs:

- `k_search_efficiency`: trace-derived null model from postprocess.
- `paper_k`: paired blind baseline from basin runs with `--basin-blind`.

1.7. 7. Intervention Protocol

For each theorem, we first solve a wild-type run and extract the solution path $\pi = \{\tau_1, ..., \tau_n\}$. We then run controlled lesions by blocking one tactic (or tactic family) from that path and rerun under the same budget and configuration.

This gives a clean comparison: same theorem, same search budget, one constrained action channel, repeated across all path tactics.

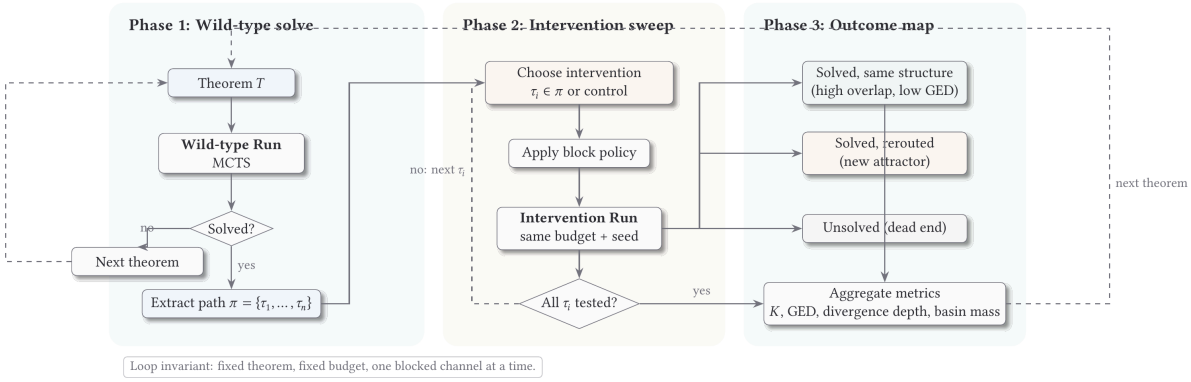

Figure 4: Canonical Loop

1.7.1. How to Read Attractor Analysis

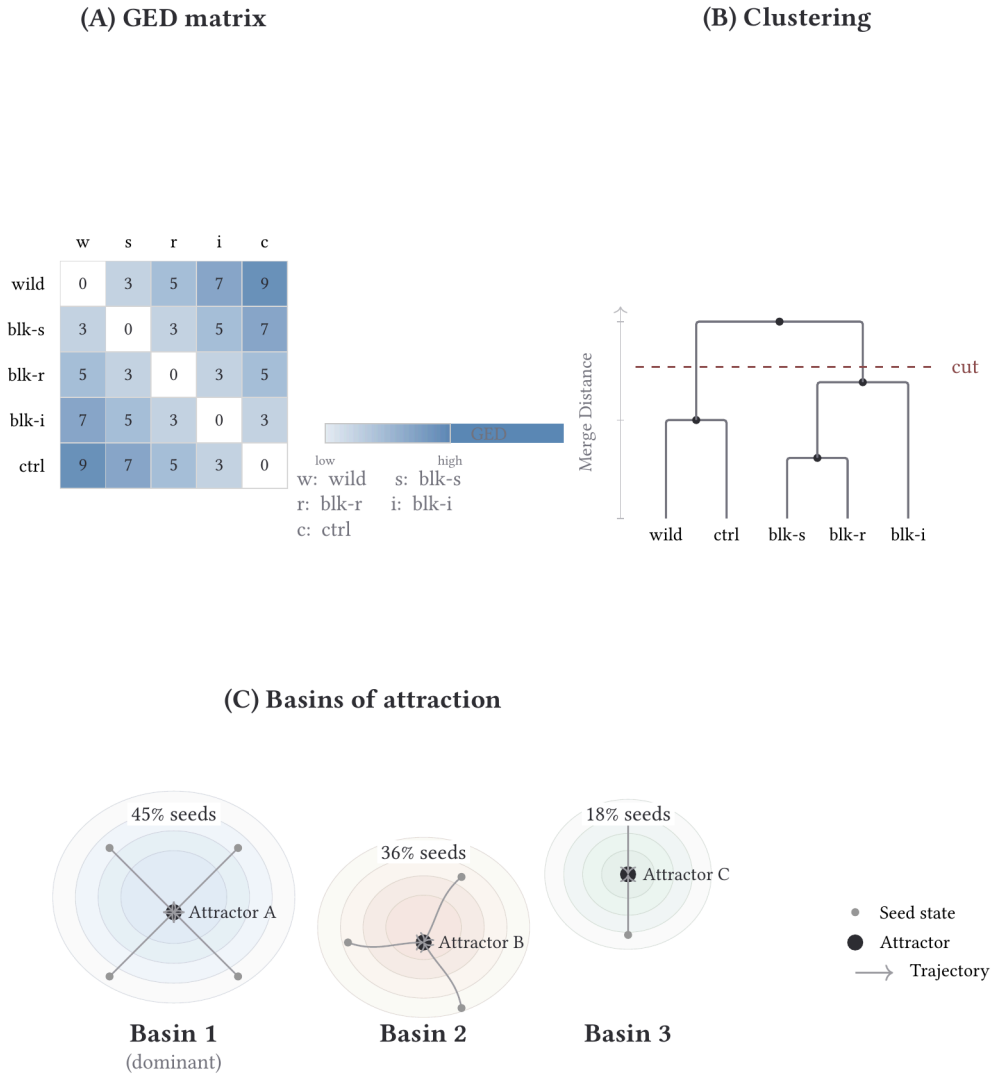

Figure 5: Attractor Analysis

- Panel A (GED matrix): pairwise structural distance between runs.
- Panel B (clustering + cut): where we place the cut determines attractor families.
- Panel C (basins): seed mass captured by each attractor family.

Interpretation: low GED + large shared basin mass implies robust proof structure; high GED with split mass implies genuine rerouting under intervention.

1.8. 8. Log-Derived Vignettes

1.8.1. Vignette Gallery Player

[Interactive element – see web version at specterlab.org/blog/wonton-soup/]