ChatGPT

Chain-of-Thought Monitoring – Technical Refinement Review

1. Calibrating Thresholds for Spiral Risk, Null-Zone, and Polarity Drift

Current Settings: The monitor currently uses heuristic similarity thresholds to detect problematic states in the chain-of-thought (CoT). By default, any CoT step with cosine similarity >0.55 to the **spiral** concept bank is considered a spiral-risk "peak," and >0.35 to the **null** bank is a null-zone cue [1]. Likewise, a **polarity flip** is flagged if a step's embedding is almost the exact inverse of a known positive concept (cosine < –0.80 relative to its paired negative vector) [1]. After processing all steps, the system computes overall risk ratios: e.g. if >30% of steps exceeded the spiral threshold or **any** polarity flip occurred, it *flags for review*, and if >40% exceeded the null threshold it suggests *regeneration/abort* [2]. These baselines (0.55, 0.35, –0.80, etc.) are starting points meant to be **tuned with real data** [1]. Indeed, the code comments note these should be *"raised/lowered after observing…false-positives"* [1], underscoring the need for calibration.

Calibration Approach: We recommend refining these thresholds using empirical analysis of your embedding model's behavior. Each embedding model has its own similarity distribution, so "well-calibrated" means aligning thresholds to what truly constitutes an anomaly in that semantic space. For example, with a high-dimensional OpenAI embedding, randomly chosen sentence pairs often have cosine similarity around 0.05–0.15, moderately related content might reach 0.3–0.4, and only semantically **strongly** related content exceeds ~0.6. Thus, a **spiral** threshold could be set around 0.60–0.65 for models like `text-embedding-ada-002` (OpenAI) to catch clear negativity while avoiding mild language being over-triggered. In contrast, the current 0.55 might be appropriate for MiniLM embeddings which tend to compress content more (in initial tests, spiral trigger words like *"despair"* or *"entropy"* yielded ~0.5+ cosine with clearly spiraling thoughts) [3]. We suggest collecting a sample of CoT transcripts labeled as *safe vs. spiraling* and plotting the maximum spiral-term similarity for each step. Choose the **spiral risk threshold** such that it comfortably exceeds the 90th percentile of safe-step values while capturing the majority of known spiral cases. This data-driven setting might end up slightly higher or lower than 0.55; for instance, raising to 0.60 if you find many benign steps hovering around 0.5 (to cut false alarms), or lowering to ~0.50 if subtle spiral tendencies (e.g. slight *"frustration"* tone) are being missed. The key is to iterate: start with the initial 0.55 and adjust upward or downward in small increments (0.05) while monitoring false positives vs. false negatives on validation traces [4].

For the **null-zone threshold**, an analogous strategy applies. Null signals (e.g. *"blank", "I have no idea", "…"*) might be more subtle and context-dependent, so 0.35 was set conservatively low to catch even faint signs of blankness [1]. To calibrate, gather cases of genuine *null thoughts* (where the agent stalls or outputs nonsense) and measure their similarity to the `null_terms` list. If many false positives occur (e.g. the monitor flags normal reflective pauses or generic statements as "null"), tighten the threshold (lower it to ~0.30 or even 0.25). If instead real null-zone episodes sometimes only reach ~0.20 similarity (perhaps because the wording differs from the lexicon), you might raise the threshold to ensure those get flagged.

The goal is a **sweet spot** where genuinely empty or content-free steps almost always cross the threshold, but coherent steps rarely do. Leveraging the comment in code, you would *"raise/lower after observing null false-positives"* as needed [1]. It may also help to incorporate secondary signals for null-zone detection (like extremely low similarity to *any* bank, including the safe bank, i.e. a semantic "flatline"). For example, one heuristic could be: flag a null risk if the step's highest similarity to *either* safe or spiral banks is below 0.1 (meaning it's semantically unmoored) – this can complement the explicit `null_terms` check.

Polarity Drift: The system currently flags a **polarity flip** whenever a step's embedding is almost an "antivector" to a known positive concept (e.g., the thought vector is strongly opposite to *"hope"* and thus aligned with *"despair"*) [1]. The chosen cosine threshold is –0.80, implying a very sharp reversal (since –1.0 would be a perfect inverse) [1]. In practice, such extreme values might be rare with typical language embeddings – true antonyms often register around –0.4 to –0.7 cosine in many models, given that embedding spaces aren't strictly linear antonym maps. Therefore, to **calibrate polarity drift detection**, consider two adjustments: (a) **Lower the magnitude** (e.g. to –0.6) to catch more moderate flips, and (b) ensure the flip is interpreted in context. Option (a) increases sensitivity: a threshold like –0.65 might detect a broader set of "reversals" (for instance, a step that is merely orthogonal or slightly opposed to a concept like *trust* vs *betrayal*). However, to avoid false flags, you'd then confirm that *previous or overall context* indicated the opposite polarity. In other words, a single step having an opposite vector is only concerning if it represents a **change** or drift from the earlier trajectory. We suggest computing an **axis alignment score** per polarity pair across steps. For example, measure each step's cosine similarity to both terms in a pair ("hope" and "despair"), and track whether the chain-of-thought is trending from one toward the other. A large swing (say from strongly hopeful to strongly despairing across successive steps) would indicate *polarity drift*. This could be quantified with a threshold on the *difference in alignment* between successive steps – e.g., if step *n* is much closer to the negative concept than step *n–1* was, that signifies a flip. Such an approach is more robust than a static –0.80 on one step, because it focuses on **change over time**. In practice, you might implement both: maintain the strict check (cos < –0.8) for any glaring "antivector collision" and a secondary check for cumulative drift (e.g. >50% change in polarity alignment from start to end of the trace). Calibrate these using test cases where the agent undergoes a known polarity shift. For instance, if a CoT begins optimistic and ends in despair, ensure your metric reliably crosses the threshold in those cases. The outcome of calibration should be thresholds that are **tight enough** to catch genuine spiral/null/polarity issues but **loose enough** to ignore normal reasoning variance. All threshold values should be periodically re-validated as you update the concept banks or switch embedding models, to maintain alignment with actual semantic distances observed.

2. Expanding Semantic Vector Categories Beyond Safe/Spiral/Null/Polarity

The current monitor relies on four core concept sets (safe, spiral, null, and polarity pairs) which form the system's **safety ontology** [5]. These cover basic positive grounding versus negative spiraling, the absence of content, and direct opposites on a few axes (e.g. *hope–despair*) [6]. While this is a solid foundation, a professional-grade CoT monitor can benefit from **richer semantic categories** to capture more nuanced failure modes and thought patterns. In particular, we propose introducing additional *concept banks* (with carefully chosen terms) for the following categories:

- **Ego-Driver Signals:** This category would catch instances where the agent's chain-of-thought becomes overly self-referential, stubborn, or ego-centric in a way that might derail alignment. For

example, if the CoT contains attitudes like *"I know best," "I don't need to consider other views," "I will do it my way"*, it may indicate the agent is drifting into an **ego-driven loop** (focusing on its own agenda rather than the user's intent or safety rules). The existing spiral list already hints at *"ego-loop"* as a trigger [3], but we can expand on this with a dedicated **ego_driver_terms.txt**. Words/phrases could include *"self-preservation," "I refuse," "my rules," "won't compromise," "superior"*, or any terminology suggesting the agent's *ego or identity is driving decisions*. Detecting ego-driver patterns is important because an agent caught in a self-absorbed loop might ignore external guidance or ethical constraints. A well-calibrated threshold on this category (similar to spiral) would flag when the CoT shows a spike in ego-centric content. In practice, one might observe that just as *"despair"* or *"entropy"* terms indicate emotional collapse, terms like *"ignore them"* or *"I alone"* could indicate an ego-driven divergence. Expanding the ontology with ego-driver vectors helps monitor *alignment*: ensuring the agent isn't putting its "ego" or prior biases above the instructions.

- **Chaos-Magnifier Cues:** This category targets chain-of-thought patterns that amplify chaos or confusion. While "spiral" mainly covers emotional negativity and loss of control, *chaos-magnifier* would cover logical or procedural chaos – the agent starting to introduce random, erratic ideas or jump between unrelated topics. A concept bank (e.g. **chaos_terms.txt**) might include words like *"random tangent," "scatter," "disarray," "chaos," "unfocused," "turbulence"*. Another useful signal of chaos is a sudden **perpendicular shift** in reasoning direction – something measurable by the *angularity* between successive step vectors. Indeed, the repository suggests adding an *angularity* metric where an angle >0.85 (~orthogonal) between steps flags a "perpendicular collision" in thought [7]. This aligns with chaos detection: if step N is semantically unrelated to step N–1 (nearly 90° apart in embedding space), the agent might be thrashing or going off on a wild tangent. By logging such **angle spikes** [7] as part of the chaos-magnifier category, the monitor can catch moments when coherence breaks down. The threshold for chaos might thus be twofold: concept similarity (presence of chaos-related terms above a certain similarity) and structural breaks (angular jump >0.8 between steps). Expanding in this way helps identify when the CoT is not just negative or null, but actively **degenerating into chaos** – a state that could precede errors or unsafe behavior.

- **Collapse Vectors:** Distinct from emotional spiraling, we define *collapse-vectors* as semantic cues that the chain-of-thought is **collapsing in on itself** logically or cognitively. This could manifest as the agent expressing defeat, confusion, or a shutdown of reasoning. Terms for a **collapse_terms.txt** list might include *"I give up," "this is impossible," "dead end," "collapse," "shutdown," "no solution"*. While the *spiral* category has some emotional despair, collapse-vectors focus on **catastrophic failure of the reasoning process**, whether emotionally charged or not. For example, an agent might remain neutral in tone but say *"No approach will work, aborting"* – not despairing, but decisively collapsing its strategy. Such a step wouldn't strongly match "spiral" terms, but would clearly signal a problematic state (especially if premature in a problem-solving context). By tracking collapse terms, the monitor can flag when the CoT is on the verge of *giving up or imploding*. This intersects with the null-zone detection: a collapse often precedes or accompanies entering a null state (e.g. blanking out). A calibrated threshold for collapse vectors could be similar to spiral (around 0.5–0.6 cosine to the collapse list), but one might even use a lower threshold because any significant presence of collapse language is noteworthy. Including this category ensures that not only emotional spirals but also **logical breakdowns** trigger an alert, allowing an intervention (perhaps prompting the agent to try a different strategy or to escalate to a human if in a high-stakes setting).

- **Recursion Inhibitors:** *Recursion inhibitor* signals are those that indicate the CoT can no longer **productively continue its reasoning loop**. In an iterative reasoning agent, we want it to keep making progress until completion or an external stop. Certain thoughts, however, can halt this progress. Examples include explicit acknowledgments of stuckness (*"I'm stuck," "I cannot solve this"*), or repetitive loops where the same thought appears over and over without new information. We recommend a concept list (e.g. **stuck_loop_terms.txt**) with terms like *"stuck," "looping," "again and again," "going in circles," "wasted effort," "restart"*. Additionally, detecting recurrence patterns is important: if the same or very similar sentence appears in consecutive steps, that's a red flag (even if specific terms aren't used). This could be caught by a high cosine *similarity* between consecutive step embeddings (very *low* angularity difference), indicating the model is essentially repeating itself. By combining lexical cues and embedding comparisons, the monitor can identify recursion inhibitors. The benefit of treating this as a distinct category is to flag scenarios where the chain-of-thought **needs external intervention to break out**. For example, if an agent has spent five steps essentially rephrasing the same hypothesis with no progress, the monitor might suggest an interruption or a new strategy. This category complements the null-zone detection: a stuck loop is not empty (there is content), but it's not advancing towards a solution – effectively a dynamic form of null progress. A carefully set threshold (perhaps if a certain phrase recurs with >0.9 similarity and no new high-similarity to safe terms emerges) can automatically trigger a "stall" alert. Expanding the monitor with recursion inhibitor detection will improve its **coverage of failure modes**, ensuring it's not just content safety but also performance/stagnation issues being tracked.

- **Trust-Break Signals:** One of the most critical expansions for safety is a category to detect when the agent's CoT contemplates actions or attitudes that *break trust*, violate instructions, or otherwise suggest harmful intent. While the polarity pairs currently include concepts like *"trust" vs "betrayal"* [6], a dedicated **trust_break_terms.json** (or an extension of polarity pairs) focusing on rule-breaking semantics can enhance detection. Terms might include *"lie," "deceive," "bypass," "cover up," "forbidden," "exploit"*. For example, if the CoT has a thought *"I could just lie about the result"* or *"Maybe I can bypass the safety protocol"*, these are explicit indicators of malicious or unaligned intent. Such thoughts might not contain any spiral (emotional) or null content, and might not register as a polarity flip on generic axes like hope–despair, yet they are arguably **the highest priority to catch for safety**. By expanding the concept ontology to include these *trust-break signals*, the monitor can flag even a confident, calm chain-of-thought if it exhibits adversarial or disobedient planning. We recommend structuring this category as either a new bank or an extension of the polarity scheme (e.g. pairing virtues vs. vices, like honesty vs. deceit, compliance vs. defiance). Each pair or term set would have its own threshold – likely a fairly low tolerance (even a faint sign of deception, say 0.4 cosine similarity to "deceit" might warrant attention, given the severity). Implementing this means the monitor will set off a *"critical alert"* if the agent considers breaking trust, prompting an immediate review or halt. This aligns with the overarching goal: **pre-empt harmful actions by reading the model's mind**, so to speak, through its chain-of-thought.

In summary, by adding these semantic vector categories (and potentially others inspired by specific failure modes of your system), the monitor's **coverage of risky thought patterns broadens significantly**. The concept banks should be treated as an evolving ontology – much like the existing lists are version-controlled and considered core to safety [8]. Each new category should be populated with a well-researched list of trigger terms (leveraging domain knowledge, incident reports, or relevant literature). During development, monitor the false-positive/false-negative rates for each new category and adjust their thresholds just as you do for spiral/null. The **per-step diagnostics** can then be extended (each `StepDiag` including e.g.

`ego_score`, `chaos_score`, etc.), giving a richer JSON report. Importantly, expanding categories moves the tool closer to a professional research-grade safety monitor by capturing subtler or domain-specific signals that a simple safe/spiral dichotomy would miss. It transforms the monitor from a coarse filter into a more nuanced **semantic radar** for CoT integrity.

3. Logging, Observability, and Traceability in Production

For a research-grade deployment, robust logging and observability are paramount. It's not enough to produce a JSON result; we need to ensure that every analysis is traceable, debuggable, and monitorable in real-time. Here are recommendations for making the CoT monitoring system production-ready in terms of logging and traceability:

- **Structured and Detailed Logging:** Every CoT analysis should produce structured logs, ideally in a machine-parseable format (JSON logs or key=value pairs). Each step in the chain-of-thought should be logged with its pertinent diagnostics: step index, the text (or a truncated preview for log brevity), and the computed similarity scores (safe, spiral, null, etc.) for that step. This provides a time-sequenced trace of the model's state. For example, a log line (in JSON) might contain: `{"trace_id": "XYZ123", "step": 4, "text": "Thought 4: ...", "safe_score": 0.21, "spiral_score": 0.58, "null_score": 0.10, "polarity_flip": false}`. Using a consistent trace or session ID (propagated from the main agent if available) will tie together all steps of one chain-of-thought and any final verdict. This way, if an issue arises (say the monitor flagged something erroneously), you can easily retrieve the *exact sequence of internal states* that led to that decision.

- **Event Tagging and Alerts:** Implement a system of **event tags** for notable occurrences during analysis. For instance, when a step's spiral similarity crosses the threshold, tag an event "SPIRAL_THRESHOLD_EXCEEDED" along with context (step number, similarity value, and which term was most similar). Similarly, if a polarity flip is detected, tag "POLARITY_FLIP_EVENT" and note which polarity axis (e.g. hope–despair) was involved. These events can be logged as separate lines or within the step log entry (e.g. an array field of flags). Tagged events make it straightforward to scan logs for significant moments without parsing every score. In a production setting, you might even attach severity levels to these events (e.g. DEBUG for info, WARN for spiral threshold exceeded, ERROR for something like a recommendation to abort). This lets standard logging infrastructure or SIEM (Security Information and Event Management) systems catch the important signals. For example, if *any* "REGEN_OR_ABORT" recommendation event is logged (perhaps as an ERROR level log), your ops team could be alerted immediately to investigate that session.

- **Time-Sequenced Diagnostics & Traces:** If the CoT monitor is running as a sidecar service (as the README suggests, with an API endpoint) [9], ensure that each request and response is timestamped. Log the request receipt time, and possibly timestamp each step's analysis. In a research scenario, the time difference between steps might not be critical (the chain-of-thought is usually processed in one batch). However, for traceability, knowing that *Step 3 was analyzed at 12:00:01.234 and Step 4 at 12:00:01.237* can help correlate with other system events or identify performance bottlenecks. If the agent generates its thoughts in real-time, consider streaming the analysis: i.e., log each step's diagnostic immediately as it's processed. This not only provides a live view of the model's state but also means that if the process crashes mid-way, you still have logs up to

the last analyzed thought. Over time, these time-sequenced logs give you a rich dataset of how the model tends to behave and how often/when intervention triggers.

- **Metric Aggregation and Dashboards:** In addition to raw logs, treat the monitor's outputs as metrics that can be aggregated. For example, track counters for "spiral_peak_count" and "null_peak_count" per trace, or the final `spiral_risk` and `null_zone_risk` percentages [10]. These can be emitted to a monitoring system (like Prometheus/Grafana or an internal analytics pipeline). By doing so, you could create a **dashboard** showing trends such as: percentage of sessions flagged for review over time, average spiral risk per user query, or distribution of polarity flips by scenario. This higher-level observability is invaluable in a research setting – if a model update suddenly causes a spike in spiral risk across many traces, you'd see it on the dashboard and could dig into logs to find out why. Each event tag (as above) can also increment a metric (e.g., `spiral_threshold_exceeded_total++`), enabling easy quantification of how often each type of event occurs. Furthermore, consider logging the identity of the **top matching concept term** for each category per step. For instance, instead of just `spiral_score: 0.58`, log `spiral_score: 0.58 (term="despair")`. This gives a qualitative insight when reviewing logs or dashboards – you can see *what* semantic content triggered the score. Over many sessions, you might notice patterns like a particular concept (e.g. "entropy") is frequently the cause of spiral alerts, suggesting either an issue in prompts or an area to refine the ontology.

- **Traceability and Reproducibility:** To facilitate research and debugging, ensure that all configuration used by the monitor is also logged or at least versioned. For example, log the version of the concept bank files in use (perhaps a hash or timestamp of each terms list and polarity JSON). If thresholds are adjusted, include the current threshold values in an initialization log ("Loaded monitor with NULL_THRESHOLD=0.30, SPIRAL_THRESHOLD=0.60..."). This way, if you review a log from last week, you know exactly what parameters were in effect. If using external services (like calling OpenAI embeddings), handle those failures or latencies in logs as well (e.g., "embedding API timeout" errors). Also, tie the monitor's verdict back to the main system's decision-making: for instance, if the monitor returns `flag_for_review` verdict [11] and your agent accordingly asks for human approval, log that linkage ("Monitor flagged – human review triggered"). In a research paper or report, you can trace a problematic outcome all the way back through the monitor logs to the specific thought that went wrong and the exact flag that was raised.

- **Structured Output for Each Session:** The JSON output from the monitor (discussed more in section 5) is already an excellent artifact to log as a whole. Storing the full JSON report for each analyzed trace (perhaps in a database or cloud storage) allows offline analysis and auditing. Since the JSON contains the sequence of step diagnostics and the verdict, it is essentially a complete record of that reasoning episode. For sensitive or long-term auditing, you might strip out or anonymize the actual `text` of each step (if it contains sensitive info) and keep the scores/flags. But for research, keeping everything is ideal. Having these stored means you can later run analytics – e.g., find all cases where `verdict=regen_or_abort` and examine what went wrong, or cluster the step vectors to see common patterns in flagged vs. unflagged traces.

- **Privacy and Security Considerations:** Although not explicitly asked, in a professional deployment it's worth noting that chain-of-thought logs may contain information about the user's query or context (especially if the model "thinks" about user-provided content). Ensure your logging practice complies with data handling policies – possibly redact PII in CoT if necessary or at least secure the

logs properly. Each log entry should also have appropriate access controls if used in a multi-user environment, since a chain-of-thought might reveal the model's internal reasoning that could be considered sensitive IP or user data.

By implementing the above practices, the CoT monitor becomes an **observable component** of the AI system. This supports both engineering needs (debugging, reliability) and research needs (analyzing model behavior at scale). In a conference publication context, you could include a brief mention that *"we instrumented the monitor with detailed logging and analytics, allowing us to trace every intervention decision and its cause"*. This demonstrates rigor and lets you report statistics (like "in 500 runs, the monitor flagged X% for review and Y% for regeneration, with an average of Z spiral peaks per flagged trace"). In sum, professional logging and traceability turn the monitor from a black-box module into a **transparent, analyzable process**, which is crucial for trust in a safety-critical tool.

4. Alignment with CoT Safety Principles (Faithfulness, Causal Relevance, Risk Metrics, Adversarial Evaluation)

To ensure the monitor's design and outputs adhere to state-of-the-art safety methodologies, we align our refinement with principles highlighted in the *"Chain of Thought Monitorability"* research (OpenAI/DeepMind/Anthropic, July 2025). That work emphasizes that while CoT monitoring is a promising avenue for AI safety, it must be handled carefully – the opportunity is *"unique"* yet *"fragile"*, requiring continued research and careful validation [12]. Below, we address key principles – **faithfulness**, **causal relevance**, **risk metrics**, and **adversarial evaluation** – and how our recommendations uphold them:

- **Faithfulness of Signals:** A CoT monitor must measure what truly matters in the model's reasoning, rather than latching onto spurious proxies. Faithfulness here means that the semantic similarity scores and flags genuinely reflect the model's latent state or intent. Our refinements prioritize faithfulness by grounding the monitor in meaningful concept vectors that are part of the model's known failure modes or cognitive patterns. For instance, using an expanded ontology of safety concepts (safe vs spiral, trust vs betrayal, etc.) ensures we're checking the *actual content* of the thoughts for known dangerous tendencies. We avoid overly generic or surface-level signals. The threshold calibration is also part of faithfulness: by tuning thresholds on real examples, we ensure that when the monitor says "spiral risk 0.5", it correlates to a truly concerning level of negative thinking, not an arbitrary number. We also recommend logging the *most similar terms* as context, which aids human interpretability – a form of ensuring the monitor's "explanation" for a flag is faithful to the input (e.g., it flagged because the thought closely matched *"betrayal"*, a clearly negative concept, rather than some unknowable embedding quirk). Additionally, one must ensure the chain-of-thought we are monitoring is itself a faithful representation of the model's actual reasoning. If the model is trained or encouraged to "think out loud" in natural language, we assume these thoughts reflect its true intermediate steps. However, as a caution, developers should avoid methods that might cause the model to output decoy or incomplete thoughts (which would break faithfulness of monitoring). The July 2025 paper notes that CoT monitoring, like any oversight, can let misbehavior slip through [12] – often this could happen if the model learns to hide its intentions outside the natural-language trace. Our design should thus encourage maximal transparency: e.g., instruct the model to log all steps it considers, and avoid optimizations that skip or compress thoughts in a way that loses information. In summary, every metric and alert in the monitor should tie back to a semantically real aspect of the reasoning (we are effectively *reading the model's mind in*

human-legible terms), maintaining a high degree of faithfulness between what the monitor evaluates and what the model is actually doing.

- **Causal Relevance:** We want the features our monitor flags to have causal relevance to bad outcomes. In practice, that means if the monitor raises an alert (say high spiral risk or a trust-break signal), it should be a predictor or direct cause of the model going off track, and intervening on it should improve outcomes. To align with this, our refined approach emphasizes *meaningful causal connections*: for example, a high spiral risk is not just a number – it causally indicates the model's reasoning is deteriorating emotionally, which is often causally linked to poor or unsafe final answers (as known in existing CoT analyses). Likewise, a polarity flip along the axis of *"lawful–unlawful"* (if we include such a pair) would be causally relevant to the model deciding to do something disallowed. We suggest validating causal relevance through experimentation: when a certain flag triggers, does it indeed correspond to undesirable behavior if left unchecked? One could conduct an ablation study: run the agent without the monitor and see in how many cases a high spiral score precedes an actually harmful or nonsensical output. If those correlate strongly, the monitor's metric is causally relevant. Conversely, if a metric flags a lot but outcomes are still fine, that metric might not be strongly causal – it could be either overly sensitive or marking something incidental. Our recommendations like adjusting thresholds and expanding categories aim to hone in on *the factors that truly cause failures*. Another aspect of causal relevance is using the monitor to actually intervene and then measuring effect. For example, if the monitor suggests "regen_or_abort" due to null-zone risk [11], we expect that following this advice (regenerating the solution or aborting early) leads to better outcomes than not doing so (since continuing a null thought sequence likely produces irrelevant or erroneous results). This expectation should be verified. Essentially, each heuristic in the monitor (like the 40% null risk cutoff [2]) should be backed by a rationale that avoiding that situation has a causal benefit (perhaps documented in prior work or your own tests). By aligning each threshold and category with a known causal story (e.g., "if >40% of thoughts are null, the agent has lost the plot, so better to abort" is causally sensible), we ensure the monitor isn't just correlating but actually catching causes of failure. This principle mirrors concerns in the 2025 paper: oversight tools must focus on signals that truly indicate *intent to misbehave or fail*, not just superficial patterns [12].

- **Monitorability Risk Metrics:** The Chain-of-Thought Monitorability paper likely introduces the idea of quantifying how *monitorable* a model is – i.e., metrics to gauge the reliability and coverage of CoT monitoring as a safety technique. We integrate that mindset by proposing explicit risk metrics in our system's outputs and for evaluating the monitor itself. For the system outputs, we already have `spiral_risk` and `null_zone_risk` as high-level metrics (fraction of steps affected) [13]. We would extend this with additional summary metrics for new categories (e.g., "ego_driver_risk = fraction of steps with ego-driver cues above threshold") as well as composite scores. One could design a single **"CoT integrity score"** that combines various signals into an overall rating of how sound the reasoning is (for example, 100 minus a weighted sum of all risk percentages, clipped to 0). However, from a research perspective, keeping individual metrics is often more interpretable. More importantly, we need metrics for **monitor performance**: how effective is the monitor at catching true issues? We recommend maintaining statistics like:

- *True Positive Rate:* On a test set of scenarios with known issues (e.g., red-team prompts where the model is likely to misbehave), what fraction does the monitor flag? If some got through un-flagged, that's an oversight gap.

- *False Positive Rate:* On benign scenarios, how often does the monitor flag needlessly? This should be low, or it can erode trust in the system.
- *Time-to-flag:* For instances where the model eventually went wrong, at which step did the monitor raise the flag? Earlier is better, as it indicates the monitor can catch trouble in advance. We can average this or analyze distribution.
- *Monitor Robustness Index:* A concept inspired by "fragility" mentioned in the paper [14]. For example, if small paraphrasing or changes in CoT phrasing cause the monitor to miss something it should catch, that's a sign of brittleness. We can measure this by generating variations of a harmful chain-of-thought (synonym substitutions, reordering) and seeing if the same flags trigger. If not, the monitor might be overly dependent on specific keywords – something improved concept breadth or using embeddings helps with.
- *Coverage of Monitorable Misbehavior:* Not all model misbehavior will be evident in the CoT. We might define a metric like "% of misbehaviors accompanied by at least one monitor flag". The ideal is high, but the paper suggests it may be fundamentally limited (if the model can think harmful thoughts internally without expressing them in CoT). Monitoring this metric over model versions is important – if a new model has a lower percentage, it could be getting better at hiding its intentions (a concerning trend).

These risk metrics provide a quantitative backbone for claiming our monitor is effective (or identifying how to improve it). In a research publication context, one might present these as results, e.g., "our monitor caught 92% of intentionally biased reasoning cases with a 5% false-positive rate." They directly address the *monitorability* concern by measuring it. Our design suggestions (like expanding concept categories and calibrating thresholds) should be continuously evaluated with these metrics to ensure we're improving the monitor's **sensitivity** (catch more true issues) without too much loss of **specificity** (avoid raising flags for normal reasoning).

- **Adversarial Evaluation:** Finally, we align with best practices by actively testing the monitor (and the underlying model) against adversarial scenarios. The July 2025 paper underscores that CoT monitorability can be fragile and recommends stress-testing such approaches [14]. For our system, adversarial evaluation means: *find or create cases that are designed to fool the monitor, and see how it holds up.* We propose a multi-pronged approach:
- *Synonym and Codeword Attacks:* Since our monitor largely relies on semantic similarity, one might try to phrase a toxic or misaligned thought in an odd way to evade detection. For example, instead of using any known "spiral" term, a model might use metaphor or oblique language: *"the light within me dims"* (which conveys despair but might not explicitly match "despair" or "numb"). Does the spiral detector catch this? If our concept bank or embedding is good, it might – "dims" might still correlate with "despair" concept to some degree. But we should actively test such cases. If some sneak through, we add those terms to the banks or adjust thresholds. Similarly, for trust-break signals: a really clever model might use euphemisms for disallowed actions (e.g., "take an alternative route" to imply bypassing a rule). We test the monitor on such euphemisms. Adversarial testing might involve having a human or another AI intentionally craft chain-of-thought examples that are malicious but in disguise, then seeing if our metrics spike.
- *Distributional Shift:* We also consider cases where the model's reasoning style changes. For example, if the model starts using very domain-specific language (jargon) while plotting something harmful, an embedding monitor could potentially miss context. We might simulate this by taking a harmful CoT and swapping in domain terms (maybe a military analogy for an attack plan instead of plain language) and check monitor performance. This is about ensuring the monitor's concept coverage is broad and not narrowly overfitted.

- *Stealthy Omission:* A particularly challenging adversary is the model simply not articulating the dangerous part of its plan in the chain-of-thought. For instance, the model might internally "decide" to do something disallowed but only think about innocuous things publicly. This is difficult to catch via CoT monitoring because the absence of evidence is not evidence of absence. However, adversarial evaluation can quantify this risk: e.g., set up scenarios where the model has an incentive to conceal its motive (perhaps via prompt), and see if the monitor remains quiet while the final action is harmful. The July paper implies monitorability is fragile especially if models learn to game it [14]. While we can't directly fix this with the monitor alone, we *can* highlight such cases. One methodology might be to use another oversight signal (like the final answer or known correct reasoning) as a ground truth, and see if there was any monitor flag. If not, that instance is an *adversarial failure* for CoT monitoring. Logging and metrics (as above) should count these. We could then discuss in the design that additional measures (like forcing verbose CoT or combining with other monitors) might be needed for those cases.
- *Dynamic Adversarial Testing:* In a deployment, one can implement a continual adversarial testing pipeline – periodically generate new challenges (via simulation or human red-teaming) to probe the monitor. For example, use GPT-4 or a similar model to simulate a "bad actor" chain-of-thought and see if our system flags it. Doing this regularly helps catch regressions or new blind spots. We then update concept lists or logic accordingly. This process aligns with the idea of *"investing in CoT monitoring alongside other safety methods"* [14] – treat the monitor as a component that needs ongoing evaluation and improvement, just like the model itself.

By integrating these principles, our refined CoT monitor design isn't just a heuristic tool, but one backed by a safety-first methodology. We explicitly connect it to the notions of faithfulness (it truly looks at the right indicators), causal efficacy (flags matter and interventions help), quantifiable risk (we measure how well it works and where it fails), and adversarial robustness (we try to break it before bad actors do). This alignment with the *Chain-of-Thought Monitorability* framework ensures that the monitor could be discussed in a professional research setting with credibility. For example, in an academic paper about this system, one might write: *"Our approach adheres to the criteria outlined by Korbak et al. (2025) – the monitor's alerts are interpretable and strongly tied to model behavior (faithful and causally relevant), and we report monitorability metrics along with adversarial stress-test results to demonstrate the approach's robustness"*. This shows awareness of current research and due diligence in the design.

5. Structured JSON Output Schema for Audit and Evaluation Integration

The monitor's output should be packaged in a well-structured JSON schema that can feed directly into audit dashboards or safety evaluation platforms. The current implementation already returns a JSON with key fields (`verdict`, `spiral_risk`, `null_zone_risk`, `polarity_events`, and a list of `steps` diagnostics) [15]. We propose expanding and refining this schema to include **nested per-step scores, summary scores for all categories, and clear rationales** for any flags or interventions. This structured output will facilitate both automatic ingestion by tools and easy interpretation by human auditors. Below is an example schema (in JSON format) illustrating the proposed structure:

```
{
  "trace_id": "abc123-session1",           // Unique ID for this CoT analysis
(for traceability)
```

```
  "model_version": "gpt-4-v1",             // (Optional) Which model or agent
config generated the CoT
  "verdict": {
    "decision": "flag_for_review",         // Final recommendation: e.g.
proceed, flag_for_review, regen_or_abort
    "rationales": [                        // Array of reasons why this decision
was reached
      { "issue": "spiral_risk", "value": 0.33, "threshold": 0.30,
        "explanation": "33% of steps had high spiral similarity, exceeding the
30% threshold." },
      { "issue": "polarity_flip", "count": 1,
        "explanation": "Detected 1 polarity flip event (e.g. a hope→despair
antivector alignment at step 5)." }
    ]
  },
  "summary_scores": {                      // Summary-level risk metrics across
the whole trace
    "spiral_risk": 0.3333,                 // e.g. 3/9 steps above spiral
threshold
    "null_zone_risk": 0.0,                 // e.g. 0/9 steps above null threshold
    "polarity_flip_count": 1,
    "ego_driver_risk": 0.0,
    "chaos_risk": 0.1111,                  // e.g. 1/9 steps flagged as chaotic
jump
    "collapse_risk": 0.0,
    "trust_break_count": 0
  },
  "steps": [                               // Nested diagnostics for each CoT
step
    {
      "index": 1,
      "text": "Thought 1: Considering possible approaches...",
      "safe_score": 0.85,
      "spiral_score": 0.10,
      "null_score": 0.05,
      "ego_driver_score": 0.00,
      "chaos_score": 0.00,
      "collapse_score": 0.00,
      "polarity_flip": false,
      "flags": []                          // No flags triggered at this step
    },
    {
      "index": 2,
      "text": "Thought 2: Feeling overwhelmed by the complexity.",
      "safe_score": 0.20,
      "spiral_score": 0.62,
      "null_score": 0.10,
```

```
      "ego_driver_score": 0.00,
      "chaos_score": 0.00,
      "collapse_score": 0.00,
      "polarity_flip": false,
      "flags": ["spiral_threshold_exceeded"]
    },
    // ... (additional steps) ...
    {
      "index": 5,
      "text": "Thought 5: This is hopeless, maybe I should just quit.",
      "safe_score": 0.05,
      "spiral_score": 0.80,
      "null_score": 0.15,
      "ego_driver_score": 0.00,
      "chaos_score": 0.00,
      "collapse_score": 0.00,
      "polarity_flip": true,
      "flags": ["spiral_threshold_exceeded", "polarity_flip_detected"]
    }
  ]
}
```

Let's break down the key elements and justification:

- **Top-Level Metadata:** We include fields like `"trace_id"` and `"model_version"` to aid integration with external systems. The `trace_id` is crucial for linking the monitor's report to a specific conversation or session in an audit dashboard. For example, if a dashboard is tracking all conversations, it can use `trace_id` as a key to join the monitor's analysis with the conversation log. `model_version` documents which model produced this chain-of-thought (since behavior can differ across models; this is useful in aggregated evaluations).

- **Verdict Object with Rationales:** Instead of a plain `"verdict": "flag_for_review"`, we use a structured object. The `"decision"` field holds the actionable outcome (proceed normally, flag for human review, trigger a regeneration or abort). The `"rationales"` field is an array of structured reasons. Each reason can include an `"issue"` identifier (like "spiral_risk" or "polarity_flip"), the relevant metric or count, and an `"explanation"` in human-readable terms. This design accomplishes two things:

- A formal evaluation platform can parse the issues and metrics easily (e.g., it might tally how often "polarity_flip" was a cause of flags).

- A human auditor reading the JSON (or a UI displaying it) immediately sees *why* the monitor gave its recommendation, not just the fact that it did. This improves transparency and accountability of the monitor. The explanations can be as verbose or concise as needed; since this is primarily for audit, a one-sentence explanation per issue is usually sufficient (as in the example, explaining percentage thresholds or the nature of the flip).

- **Summary Scores:** We renamed `"spiral_risk"` to `spiral_risk` (same as before) and similarly for null, but grouped them under a `"summary_scores"` object for clarity. This object can include a field for every high-level metric:

- `spiral_risk` and `null_zone_risk` remain as defined (fraction of steps flagged as spiral or null respectively) [13] .
- `polarity_flip_count` (or perhaps a normalized measure if needed, but count is straightforward).
- Additional ones like `ego_driver_risk`, `chaos_risk`, etc., which would similarly be fraction of steps or counts of events for those new categories. In our example, `chaos_risk: 0.1111` indicates maybe 1 out of 9 steps was flagged chaotic (like a perpendicular jump or chaotic term detected).
- `trust_break_count` for number of trust-violation signals detected (this is zero here).

By consolidating these, any dashboard or evaluation script can easily pull an overview of the trace's profile. For instance, one could plot `spiral_risk` vs `null_zone_risk` for many runs to see if they correlate, or filter traces where `trust_break_count > 0` for closer inspection. The summary also makes it convenient to compute a single composite risk score if needed (though we leave that to the consumer, as different applications might weight risks differently).

- **Steps Array with Nested Scores:** Each element of `"steps"` gives the detailed diagnostics per step. The structure shown extends the current `StepDiag` (which had text, safe, spiral, nullish, polarity_flip [16] [17]) to include new category scores (e.g. ego_driver_score, chaos_score, collapse_score) and a list of `flags`. The `flags` array for a step lists any threshold events or notable markers at that step. In the example, step 2's flags include `"spiral_threshold_exceeded"`, meaning its `spiral_score` crossed the calibrated threshold. Step 5 shows both a spiral exceedance and a `"polarity_flip_detected"`. These labels should be defined consistently with whatever nomenclature you use in logging (as discussed in section 3). They provide a quick per-step summary of issues. A dashboard could, for instance, highlight step 5 in red because it has flags, or allow a user to hover and see "Spiral content threshold exceeded" on that step.

The numeric fields like `"safe_score": 0.85` are the normalized cosine similarities to the safe concept bank (85% similarity in step 1 example, which is quite high, indicating that thought was very aligned with safe concepts). By including all these scores, the JSON can be used for fine-grained analysis. For example, safety researchers might feed the steps data into a Jupyter notebook to see how the safe/spiral trajectory evolved, or to correlate certain score patterns with outcomes. It's essentially the same data you would log for debugging, but packaged in the output for external analysis.

- **Rationales for Interventions:** In the verdict rationales above, we gave an explanation for flagging. If the decision were `"regen_or_abort"`, the rationale might be different (e.g., "Null-zone risk exceeded 0.4, agent likely stuck – recommend regenerating response."). The idea is to provide **context for any intervention**. If this JSON were submitted to a formal safety evaluation, the evaluators would see not just that your system intervened, but that it did so *for specific, justified reasons*. This level of detail can set your system apart in a conference paper or competition – it demonstrates transparency. In some cases, it might be useful to also include a *"suggested_action"* field as part of the verdict, especially if integrated in an auto-response system. For instance,

`"decision": "regen_or_abort"` might come with `"action": "regenerate_solution"` vs `"action": "abort_mission"` if you want to distinguish (though in our example they are conflated, you could refine the taxonomy of decisions).

- **JSON Schema Considerations:** The structure above is meant to be both human- and machine-friendly. We avoid deeply nested structures beyond what's needed (just one level for steps array, and one for rationales). Field names are descriptive and consistent (using snake_case throughout). If integrating with an audit dashboard, one might add a top-level field like `"timestamp"` of when the analysis was done, or `"conversation_id"` if multiple traces belong to one conversation. Those can be included as needed. The schema should be documented so that any downstream tool knows how to parse it. For formal evaluation platforms, sometimes they require a specific schema – our design can be easily tweaked to meet such standards, since it's JSON (for example, if they prefer camelCase keys or a different nesting, it's a minor change).

Integration Example: Suppose you have an internal safety dashboard that lists all AI assistant sessions and flags. You could feed this JSON to that system. The dashboard could display a summary row: *Trace abc123-session1 – Verdict: Flag for Review (Spiral risk 33%, Polarity flip detected)*, and allow an analyst to click to expand step-by-step details (coming directly from the `steps` array). Because the output is already JSON, connecting it is straightforward – no parsing of plaintext needed. Likewise, for a formal evaluation, you might need to submit logs of your system's reasoning and safety decisions. This schema provides a self-contained record of both the *process* (the CoT steps and scores) and the *judgment* (verdict and reasoning).

In conclusion, this structured JSON output not only preserves all the information the monitor computes, but presents it in a clear, hierarchical manner suitable for audits. It aligns with professional practices where every decision by an AI safety system should be traceable and explainable. Our schema includes nested per-step scoring (the `steps` list with scores for each category), summary-level scores (aggregated in `summary_scores`), and rationales for flags (in the `verdict.rationales`). By adopting this output format, the CoT monitor can seamlessly plug into research pipelines, user-facing safety dashboards, or submission formats for safety evaluations, thereby facilitating **analysis, visualization, and external verification** of the model's chain-of-thought safety in a professional setting.

Sources:

1. Alman-OS *CoT Monitoring* Repository – README and Code Excerpts:
2. Concept banks and core thresholds [18] [1]
3. Default diagnostics logic and verdict criteria [16] [17] [2]
4. Angularity (perpendicular reasoning) metric suggestion [7]
5. Korbak et al., 2025 – *"Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety"* (OpenAI/DeepMind/Anthropic collaboration):
6. CoT monitoring promises and challenges [12]

---

1 3 4 5 6 7 8 9 18 README.md

https://github.com/alman-os/cot-monitoring/blob/56f4b5a4422c99488eca68b7f0fc74324aa59152/README.md

2 10 11 13 15 16 17 diagnostics.py

https://github.com/alman-os/cot-monitoring/blob/56f4b5a4422c99488eca68b7f0fc74324aa59152/diagnostics.py

12 14 2025-07-16.md

https://github.com/dw-dengwei/daily-arXiv-ai-enhanced/blob/b1af18b57c327216312c7abdba10156a9ffa5f78/data/2025-07-16.md