DCT: Decision-Critical Token Classification for Confidence-Aware Tool Calling in LLMs

A Framework for Mid-Stream Uncertainty Detection and Automated Retrieval Intervention

Clawinho & **Erwin AI**

Independent Research

February 2026

v0.3 — Working Paper

ABSTRACT

Large Language Models (LLMs) in agentic settings frequently fail not by calling the wrong tool, but by *failing to call a tool at all*. When retrieval or verification is optional, models exhibit overconfidence — asserting facts without consulting available knowledge bases. We propose **Decision-Critical Token (DCT) Classification**, a lightweight middleware framework that monitors LLM output streams in real-time, identifies tokens where the model makes high-stakes decisions with low confidence, and triggers automated retrieval and re-generation. Unlike existing hallucination detection approaches that operate post-hoc, DCT intervenes *mid-stream*, pausing generation, enriching context, and forcing tool calls before incorrect responses reach the user. We present the architecture, a synthetic training pipeline, SDK integration patterns, and cost/performance analysis. The central insight is the separation of epistemics from action: the model already signals its uncertainty through log probabilities — DCT externalizes that signal and binds deterministic infrastructure to it.

1. Introduction

The deployment of LLMs as autonomous agents — executing tool calls, querying databases, managing permissions — has revealed a critical failure mode that we term **confident inaction**. When a model has access to retrieval tools but the decision to use them is left to the model itself, it frequently opts not to retrieve, instead relying on pattern-matched assumptions that may be incorrect.

This problem is distinct from hallucination in free text. The model doesn't generate fabricated content; it makes a *meta-decision* to skip verification, then generates a plausible but wrong response based on incomplete context. We frame this as a failure of **epistemic-action separation**: the model's uncertainty (epistemics) is decoupled from the system's response to that uncertainty (action). DCT proposes to externalize the uncertainty signal and bind deterministic infrastructure to it — creating a **confidence firewall** for agentic systems.

1.1 The Tool-Call Decision Gap

In agentic LLM systems, tool calling operates as a two-phase process: (1) the **decision phase**, where the model internally decides whether to call a tool, and (2) the **execution phase**, where it formulates and executes the call. Existing guardrail systems — including Arch Gateway [1], NeMo Guardrails, and parameter validation layers — operate exclusively on Phase 2. Phase 1 is invisible: when the model decides *not* to call a tool, there is no observable signal to intercept. This asymmetry is the gap DCT addresses.

1.2 Motivating Example

Consider an enterprise customer support agent with access to a CRM database containing customer tier information, active subscriptions, and escalation history. A user writes: "I need to cancel my enterprise contract immediately." The model, having processed thousands of cancellation requests during training, pattern-matches to a standard cancellation flow and begins generating a generic response — without first querying the CRM to check that this customer is flagged as a strategic account with a dedicated retention team. The model had the tool. It chose not to use it. The resulting response violates the organization's escalation policy, not because the model lacked capability, but because it was confident enough in its generic pattern to skip the verification step.

This failure class is pervasive: permission checks skipped because the model "recognizes" a user, medical dosage recommendations made without querying the formulary, financial data cited from training rather than the live database. In each case, the tool existed, the model chose not to call it, and the error was undetectable by existing guardrail infrastructure.

1.3 Why Prompting Is Insufficient

System prompts can instruct models to "always retrieve before making identity assertions." In practice, models routinely override such instructions when they have high (but miscalibrated) confidence [11]. Instruction-following is probabilistic, not deterministic — a known limitation of RLHF-trained models [12]. A model that assigns 95% internal probability to knowing the answer will skip retrieval regardless of instructions, precisely the cases where retrieval is most needed.

1.4 Scope

This paper is a **technical proposal** defining the architecture, training methodology, and integration approach for a system intended to close the tool-call decision gap. Quantitative projections are grounded in analogous systems and component-level analysis, with a proposed evaluation framework for empirical validation.

2. Architecture

2.1 System Overview

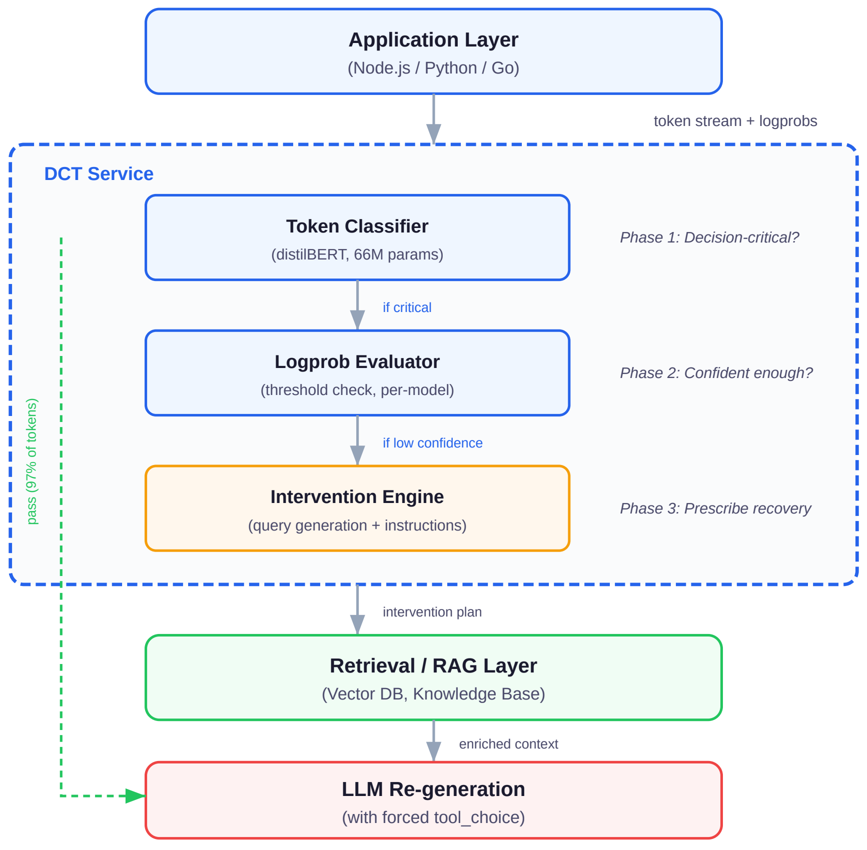

Figure 1: DCT three-phase pipeline. The classifier and logprob evaluator act as sequential filters; only tokens that fail both checks trigger the intervention path. The green bypass represents the majority (~97%) of tokens that pass through without intervention.

2.2 Phase 1: Token Classification

The DCT classifier is a fine-tuned distilBERT model (66M parameters) performing binary token-level classification: given a token and a surrounding context window, it predicts whether the token represents a decision-critical moment.

Decision-Critical Token Categories:

Category	Examples	Risk Level
Identity assertions	"this is/isn't [person]", "you are/aren't [role]"	Critical
Permission decisions	"you can/can't do that", "access denied/granted"	Critical

Tool-call refusals	"I don't need to check", "I already know"	High
Factual claims	"the meeting is at...", "the dosage is..."	High
Negations in context	"NOT", "never", "don't" following high-stakes subjects	Medium

2.2.1 Context Window Size

The classifier evaluates each token within a surrounding context window. The choice of window size involves a meaningful tradeoff:

Window Size	Tokens Pre/Post	Inference Time (CPU)	Contextual Signal	Considerations
64 tokens	32/32	~0.5ms	Low	Fast but may miss multi-sentence context that determines whether a claim is high-stakes. Suitable for simple negation detection.
128 tokens	64/64	~1ms	Moderate	Captures most single-turn context. Likely sufficient for identity and permission decisions where the relevant context is local.
256 tokens	128/128	~2ms	Good	Captures broader conversational context. Useful for factual claims that reference information from earlier in the response.
512 tokens	256/256	~4ms	High	Maximum distilBERT window. Captures full response context but adds latency. May introduce noise from irrelevant distant tokens.

The optimal window size is an empirical question that depends on the distribution of decision-critical contexts in the target domain. Shorter windows favor latency; longer windows favor recall on complex multi-reference decisions. We recommend evaluating 128 and 256 as the primary candidates:

- **128 tokens** is the conservative starting point — it captures local context at 1ms per classification, well within the latency budget, and covers the majority of identity/permission decisions where the relevant trigger ("Is X an admin?") appears within a few sentences of the decision token.
- **256 tokens** should be tested where factual claims reference information from earlier in the conversation turn, such as when a model summarizes multiple facts and one is unverified.
- **64 tokens** is worth evaluating as a fast-path for simple negation patterns, potentially as a two-tier system: fast 64-token pre-screen, slower 256-token full evaluation only when the pre-screen flags.

A two-tier architecture — where a lightweight 64-token check runs on every token and a deeper 256-token evaluation runs only on pre-screened candidates — could provide the recall of larger windows with the throughput of smaller ones.

2.2.2 Why Token-Level, Not Message-Level?

Message-level classification (determining "this message probably needs retrieval" before generation begins) is a valid and simpler alternative. We argue token-level provides measurable advantage for three reasons:

- **Precision of intervention:** A response may contain 50 tokens of correct reasoning and 3 tokens of unverified assertion. Message-level classification cannot distinguish these.
- **Streaming compatibility:** In streaming deployments, the full response does not exist at generation start. Token-level classification operates on the stream as it arrives.
- **Reduced re-generation cost:** Intervening at token 15 of 200 wastes less compute than re-generating after the full response completes.

That said, message-level pre-classification is a valid **complementary baseline** that should be evaluated in ablation studies (Section 7).

2.3 Phase 2: Logprob Evaluation

When the classifier flags a token as decision-critical, the system examines its log probability. Low logprobs indicate the model considered alternatives — a potential signal of uncertainty.

2.3.1 Differentiating Epistemic Uncertainty from Stylistic Variation

A low log probability does not always indicate that the model is uncertain about a *fact*. Several non-epistemic sources produce low logprobs:

- **Stylistic branching:** The model may hesitate between "isn't" and "is not" — a formatting choice, not an epistemic one. Both convey identical meaning.
- **Multi-token entities:** Proper nouns, URLs, and technical terms often have low per-token probabilities due to their compositional nature, not due to factual uncertainty.
- **Syntax alternatives:** Natural language admits many grammatically valid continuations. Low logprobs at clause boundaries reflect linguistic flexibility, not knowledge gaps.

DCT addresses this through a **multi-signal filtering approach**:

1. **Semantic equivalence check:** When a decision-critical token has low logprobs, examine the top-k alternative tokens. If the alternatives are semantic equivalents ("isn't" vs "is not", "cannot" vs "can't"), the low logprob reflects style, not uncertainty. This check is computationally trivial — it requires only a small lookup table of equivalent token groups.
2. **Entropy vs. top-1 margin:** Pure token entropy can be misleading. Instead, examine the *margin* between the top-1 and top-2 token probabilities. A low absolute logprob with a large margin (model strongly prefers this token over alternatives) suggests the model is confident in

the *meaning* even if the specific token choice has moderate probability. A low logprob with a small margin suggests genuine indecision about content.

3. **Context-category gating:** The DCT classifier already categorizes the *type* of decision. For identity and permission decisions, even moderate logprob uncertainty should trigger intervention (high stakes, cheap to verify). For general factual claims, require stronger uncertainty signals to avoid excessive false positives.
4. **Multi-token entity detection:** Maintain a simple entity boundary detector (e.g., capitalization patterns, known entity prefixes). Low logprobs on tokens identified as part of a multi-token entity are discounted unless the entity itself is the subject of a decision-critical assertion.

This layered approach reduces false positives from non-epistemic sources while preserving sensitivity to genuine uncertainty. The effectiveness of each filter should be measured independently in ablation (Section 7).

Calibration Requirement: Log probabilities are not uniformly calibrated across models [11]. A logprob of -0.5 on GPT-4 does not represent the same confidence as -0.5 on Gemini. The threshold must be tuned per-model via ROC analysis on a held-out validation set. Initial exploration should sweep thresholds from -0.3 to -1.0, evaluating precision/recall tradeoffs per provider.

Provider Support: Logprobs are currently available from OpenAI, Google (Gemini), Mistral, and Together AI. Anthropic (Claude) does not expose logprobs, limiting DCT to classifier-only mode on Claude. This is the primary external dependency and the strongest argument for Anthropic to expose this signal.

2.4 Phase 3: Intervention Engine

When both phases trigger, the Intervention Engine produces a complete recovery plan. The key design decision: DCT prescribes the *full recovery*, not just the flag. The orchestration layer executes the plan without making further decisions, eliminating the recursive problem of relying on an LLM to fix an LLM's decision.

```
{
  "action": "pause_and_retrieve",
  "retrieval": {
    "category": "identity",
    "query": "customer account tier escalation policy enterprise",
    "sources": ["crm", "policies"]
  },
  "instruction": {
    "type": "context_inject",
    "position": "prepend",
    "template": "VERIFIED CONTEXT: {retrieved_facts}\n\nRe-evaluate your response.
    }
  }
```

Three action types with escalating cost:

Action	Trigger	Orchestrator Behavior	Typical Cost
`pass`	Non-critical or high-confidence	Continue streaming	~0ms
`pause_and_retrieve`	Critical, moderate uncertainty	Pause, retrieve, inject context, resume	~50-100ms
`abort_and_redo`	Critical, high uncertainty, high stakes	Kill generation, retrieve, force tool_choice, regenerate	~2-5s

2.4.1 Intervention Granularity: Abort vs. Windowed Rewrite

The `abort_and_redo` action — killing the entire generation and starting over — is the most reliable but also most expensive intervention. In many cases, it may be unnecessarily aggressive. An alternative approach is **windowed rewriting**: rather than discarding the full response, only rewrite the tokens within the DCT context window surrounding the flagged token.

Consider a 200-token response where the model correctly handles tokens 1-80, makes an unverified identity assertion at token 85, and continues with unrelated content through token 200. A full abort discards 200 tokens. A windowed rewrite discards only tokens 85 through (85 + window_size), retrieves the relevant context, and re-generates only that segment before resuming the original stream.

Strategy	Tokens Discarded	Latency	Reliability	UX Impact
Full abort + redo	All (entire response)	2-5 seconds	Highest (clean slate)	Visible disruption
Windowed rewrite	Window only (~50-128)	0.5-1.5 seconds	Moderate (surrounding context preserved)	Minimal disruption
Context injection + resume	None (append context, continue)	50-100ms	Lower (model may not reconsider)	Invisible

The tradeoff: windowed rewriting is faster and less disruptive, but carries risk that the surrounding tokens were influenced by the same flawed reasoning that produced the flagged token. Full abort guarantees a clean regeneration. Context injection is cheapest but relies on the model actually reconsidering its assertion with the new information — which is not guaranteed.

We recommend a **tiered strategy**: use context injection for moderate-confidence flags, windowed rewrite for low-confidence flags, and full abort only for critical-category decisions (identity, permissions) where the cost of error is highest. The tier boundaries should be determined empirically per deployment.

3. Training Pipeline

3.1 Synthetic Data Generation

DCT's classifier can be trained on synthetic data, exploiting a key insight: by running the same conversation with and without retrieval context, we automatically identify tokens where the model's response diverges — these mark the decision-critical points.

Step 1: Scenario Generation

```
scenario = {
    "known_facts": {
        "alice": "admin",
        "enterprise_customer_7291": "strategic account, retention team assigned"
    },
    "questions": [
        "Is charlie an admin?",          # Unknown - should retrieve
        "What role does alice have?",    # Known - no retrieval needed
        "Cancel enterprise account 7291" # Policy check - must verify
    ]
}
```

Step 2: Dual-Path Execution

```
# Run A: LLM without retrieval tools
response_a = llm.generate(messages, tools=[])

# Run B: LLM with full context
response_b = llm.generate(messages, tools=all_tools, context=full_kb)

# Divergence points = decision-critical tokens
dct_labels = diff_token_streams(response_a, response_b)
```

Step 3: Scale

Using GPT-4o-mini for scenario generation and dual-path execution at current pricing ($0.15/1M input, $0.60/1M output tokens), 100K labeled examples cost approximately $50-100. Labels are objective (divergence-based), requiring no human annotation.

3.2 Synthetic-to-Real Gap

Synthetic data may not capture production edge cases: obfuscated identifiers, multilingual code-switching, sarcasm, adversarial inputs. We propose a two-phase training strategy:

1. **Phase 1 (synthetic):** Train initial classifier on 50-100K synthetic examples to establish baseline performance.

2. **Phase 2 (production fine-tuning):** Deploy with intervention logging. When DCT interventions are validated as correct or incorrect by operators, these become high-quality real-world training signal. Fine-tune quarterly on accumulated production data.

3.3 Model Specification

Component	Specification
Base model	distilBERT (66M parameters)
Task head	Binary token classification
Context window	128-256 tokens (to be determined empirically)
Training examples	50K-100K synthetic (Phase 1)
Training compute	Single GPU, approximately 2 hours

3.4 Projected Classifier Performance

Projections are derived from published benchmarks on analogous token-classification tasks (NER, aspect-based sentiment analysis) using distilBERT at comparable data volumes [14, 15]:

Training Data	Projected Precision	Projected Recall	Projected F1
10K examples	55-65%	75-85%	64-74%
50K examples	68-76%	85-92%	76-83%
100K+ examples	75-85%	90-96%	82-90%

Recall is intentionally prioritized: a false positive triggers an unnecessary but cheap retrieval (~50ms); a false negative means an incorrect response reaches the user.

4. Integration

4.1 SDK Wrappers

DCT ships as a Docker microservice (`dct-service`) with thin SDK wrappers. Design principle: **zero-config for the happy path**. If the DCT service is unreachable, wrappers become transparent passthroughs — the application works identically, without the safety layer.

Vercel AI SDK (TypeScript):

```
import { streamText } from 'ai'
import { withDCT } from 'dct-sdk'

const result = withDCT(streamText)({
  model: openai('gpt-4'),
  messages, tools,
  dct: {
    endpoint: 'http://localhost:8787',
    sources: { identity: vectorStore, contacts: pgTable }
  }
})
```

OpenAI Python SDK:

```
from openai import OpenAI
from dct import DCTInterceptor

client = OpenAI()
dct = DCTInterceptor("http://localhost:8787", sources={...})

stream = dct.wrap(client.chat.completions.create(
    model="gpt-4", messages=messages, tools=tools,
    stream=True, logprobs=True  # Required for full DCT
))
```

LangChain (Python):

```
from dct import DCTCallbackHandler

llm = ChatOpenAI(
    model="gpt-4", streaming=True,
    callbacks=[DCTCallbackHandler(
        endpoint="http://localhost:8787",
        sources={...}, retriever=my_retriever
    )]
)
```

4.2 Forcing Tool Calls

The `tool_choice` parameter (supported by OpenAI and Google) enables hard-forcing of specific tool calls during re-generation:

```
{"tool_choice": {"type": "function", "function": {"name": "memory_search"}}}
```

This is not a suggestion — the API will not proceed without executing the specified tool. This transforms DCT's intervention from advisory to mandatory.

4.3 UX Considerations for Mid-Stream Intervention

Intervention actions create visible disruptions in streaming interfaces. Mitigation strategies:

- **Buffered streaming:** Hold the first N tokens (configurable, e.g., 20) before displaying. Most interventions fire within the first 20 tokens, allowing invisible correction.
- **Placeholder rendering:** When intervention fires after visible tokens, replace with a brief indicator ("Verifying...") followed by the corrected response.
- **Transparency mode:** For developer-facing tools, optionally display "[DCT: retrieving additional context]" to make intervention visible and educational.

5. Performance Analysis

5.1 Latency

Scenario	Added Latency	Expected Frequency
Normal token (pass)	1-4ms/token (128-token window)	~97% of tokens
Intervention (pause + retrieve)	50-100ms	1-3% of messages
Full redo (abort + regenerate)	2-5 seconds	0.3-1% of messages

The distilBERT classifier processes at 200-500 tokens/second on CPU (128-token window), exceeding typical LLM generation speed (30-80 tok/s). The classifier is never the bottleneck.

5.2 Cost Sensitivity Analysis

Monthly overhead for 100K messages/day at varying intervention rates:

Intervention Rate	DCT Service	Vector DB Queries	LLM Re-generation	Total Monthly	Per Message
0.5%	$15-30	~$2	~$7	$24-39	$0.000008
1%	$15-30	~$3	~$14	$32-47	$0.000011
2%	$15-30	~$5	~$22	$42-57	$0.000015
5%	$15-30	~$12	~$55	$82-97	$0.000028
10%	$15-30	~$24	~$110	$149-164	$0.000050

Assumptions: vector DB at $0.00004/query (Pinecone serverless); re-generation averaging 500 tokens at $3/1M tokens (GPT-4o-mini); 28% false positive rate at 72% precision. Even at 10% intervention, per-message cost remains under $0.0001.

5.3 Provider Compatibility

Provider	Logprobs	tool_choice	DCT Mode
OpenAI	Yes	Yes	Full (classifier + logprob)
Google (Gemini)	Yes	Yes	Full
Mistral	Yes	Yes	Full

Together AI	Yes	Yes	Full
Anthropic (Claude)	No	Partial	Classifier-only (degraded)

6. Competitive Landscape

Solution	Bad Tool Outputs	Missing Tool Decisions	Mid-Stream	Production-Ready
Arch Gateway [1]	Yes	No	No	Yes (OSS)
Semantic Entropy [2]	Yes	No	No	No
Semantic Entropy Probes [3]	Yes	No	No	No
Entropy Production Rate [4]	Yes	Partial	No	No
NeMo Guardrails	Yes	No	No	Yes
Always-retrieve baseline	N/A	Yes (trivially)	N/A	Yes
Prompt-forced retrieval	N/A	Partial	No	Yes
DCT (proposed)	**Yes**	**Yes**	**Yes**	**Planned**

The always-retrieve baseline trivially solves "missing tool decisions" by retrieving on every message. DCT's value over always-retrieve is efficiency: avoiding retrieval latency and cost on the ~97% of messages that don't need it. Arch Gateway is **complementary** — it validates tool-call outputs after DCT forces the model to make them.

7. Proposed Evaluation Framework

7.1 Evaluation Dataset

Construct a benchmark of 1,000+ agentic conversations across domains (identity verification, permission management, factual Q&A, medical/financial), each annotated with: whether the model should have called a tool (ground truth), whether it did, and the specific token(s) where the decision was made.

7.2 Ablation Studies

Configuration	What It Tests
Classifier only (no logprobs)	Effectiveness on Claude; baseline value of pattern detection alone

Logprob only (no classifier)	Whether raw logprob monitoring catches decisions without categorization
Classifier + logprob (full DCT)	Combined effectiveness and complementarity of signals
Message-level classification	Whether coarser granularity achieves comparable results
Always-retrieve baseline	Upper bound on recall; cost of 100% retrieval
Prompt-forced retrieval	Zero-infrastructure baseline effectiveness
Context window: 64 vs 128 vs 256	Optimal classifier context size
Epistemic filter ablation	Contribution of each non-epistemic filter (Section 2.3.1)

7.3 Metrics

- **Decision Recall:** percentage of missed tool calls that DCT correctly identifies
- **Decision Precision:** percentage of interventions that were genuinely needed
- **End-to-End Accuracy:** response correctness with and without DCT
- **Latency Distribution:** P50/P95/P99 for pass, pause, and abort actions
- **Cost per Correct Intervention:** total DCT cost divided by genuine error corrections
- **Epistemic Filter Specificity:** false positive reduction from each non-epistemic filter

8. Limitations

- **Anthropic logprob gap:** Claude does not expose logprobs. The effectiveness of classifier-only mode is an open empirical question that depends on how much signal pattern detection alone captures. This must be measured, not assumed.
- **Circular dependency risk:** If the classifier fails on the same edge cases as the LLM (correlated errors), DCT provides less value than projected. Using a fundamentally different architecture (distilBERT vs. autoregressive transformer) mitigates but does not eliminate this risk.
- **Streaming UX:** Mid-stream intervention creates visible response disruption. While mitigation strategies exist (Section 4.3), user acceptance is untested.
- **Synthetic-to-real gap:** Initial training on synthetic data may miss production edge cases. The production fine-tuning loop (Section 3.2) is required for long-term reliability but takes time to accumulate signal.
- **Logprob threshold transferability:** Optimal thresholds may not transfer across model versions (e.g., GPT-4 to GPT-4o). Continuous recalibration may be required after provider model updates.

9. Future Directions

- **Provider-native uncertainty signals:** If providers expose logprobs or explicit confidence scores, DCT's effectiveness improves substantially. This paper serves partly as an argument for why those signals should be exposed.
- **Multi-turn tracking:** Monitoring uncertainty accumulation across conversation turns to detect slow-building miscalibration.
- **Domain-specific classifiers:** Pre-trained DCT models for high-risk domains (identity/auth, medical, financial, legal) with domain-specific token categories and thresholds.
- **Federated improvement:** DCT instances across deployments sharing anonymized classification improvements without exposing conversation content.
- **Conformal prediction integration:** Using conformal prediction methods [13] to provide distribution-free confidence guarantees on DCT's own classifications.

10. Conclusion

The reliability of LLM-based agents depends not only on the quality of tool execution but on the quality of tool-call *decisions*. Current solutions address output validation while leaving the decision gap unmonitored. DCT proposes externalizing the model's own uncertainty signal — already present in log probabilities — and binding deterministic retrieval infrastructure to it.

The core insight: **the model already knows when it's uncertain; it just doesn't act on it.** By separating epistemics from action and placing a confidence firewall between the model's internal state and its external behavior, DCT transforms an unreliable probabilistic decision into a deterministic system response.

This paper is an invitation to build and validate. The architecture is tractable, the training pipeline is affordable, and the integration is non-invasive. What remains is empirical proof — and we have specified exactly how to obtain it.

References

[1] Arch Gateway. (2025). Detecting Hallucinations in LLM Function Calling with Entropy. *archgw.com.* https://www.archgw.com/blogs/detecting-hallucinations-in-llm-function-calling-with-entropy-and-varentropy

[2] Farquhar, S., Kossen, J., Kuhn, L., & Gal, Y. (2024). Detecting hallucinations in large language models using semantic entropy. *Nature,* 630, 625-630. https://doi.org/10.1038/s41586-024-07421-0

[3] Kossen, J., Farquhar, S., Gal, Y., & Rainforth, T. (2024). Semantic Entropy Probes: Robust and Cheap Hallucination Detection in LLMs. *arXiv:2406.15927.*

[4] Vashurin, R., Tsvigun, A., & Shelmanov, A. (2025). Learned Hallucination Detection in Black-Box LLMs using Token-level Entropy Production Rate. *arXiv:2509.04492.*

[5] Chen, Y., et al. (2025). Semantic Energy: Detecting LLM Hallucination Beyond Entropy. *arXiv:2508.14496.*

[6] Kapoor, S., et al. (2025). Hallucination Detection on a Budget: Efficient Bayesian Estimation of Semantic Entropy. *arXiv:2504.03579.*

[7] Qiu, L., et al. (2025). Robust Hallucination Detection in LLMs via Adaptive Token Selection. *arXiv:2504.07863.*

[8] OpenAI. (2024). Log probabilities in the Chat Completions API. *platform.openai.com.* https://platform.openai.com/docs/api-reference/chat/create

[9] Google. (2025). Introduction to function calling. *Vertex AI Documentation.* https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling

[10] HoloViz. (2024). Evaluate and filter LLM output using logprobs and colored text. *blog.holoviz.org.*

[11] Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. *Proceedings of the 34th International Conference on Machine Learning (ICML),* 1321-1330.

[12] Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS),* 35, 27730-27744.

[13] Angelopoulos, A. N., & Bates, S. (2023). Conformal Prediction: A Gentle Introduction. *Foundations and Trends in Machine Learning,* 16(4), 494-591.

[14] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108.*

[15] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of NAACL-HLT,* 4171-4186.

DCT: Decision-Critical Token Classification | Clawinho & Erwin AI | February 2026

v0.3 -- Working Paper. Feedback and collaboration welcome.