International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

Deterministic Context Transaction Protocol (DCTP): A Governance Layer for Confidence-Gated, Versioned Context Exchange in Multi-Agent LLM Systems

Dinesh Surisetti
independent researcher

Abstract - Multi-agent large language model (LLM) systems increasingly coordinate several cooperating agents to complete a shared task, but conventional designs let each agent reinterpret upstream information freely, producing inconsistent field values, propagated hallucinations, and workflows that cannot be reproduced after the fact. We present the Deterministic Context Transaction Protocol (DCTP), a middleware layer in which agents exchange versioned, confidence-scored Context Transaction Objects (CTOs) rather than free-form text. Each CTO passes through a governed lifecycle - including an immutable LOCKED state reached only when a confidence threshold and dependency conditions are satisfied - and can be modified only through a validated update transaction rather than direct write access. A dedicated arbitration component resolves disagreements between agents using source-reliability and historical-accuracy weighting, and a replay component reconstructs prior executions from the preserved transaction log. We situate DCTP against four related bodies of work: classical optimistic concurrency control and workflow pre-locking in database systems; industry agent-communication standards (Anthropic's Model Context Protocol and Google's Agent2Agent protocol); a patented confidence-gated agent context space (WO2021084510A1); and concurrent 2025–2026 academic proposals for confidence-calibrated, trust-weighted multi-agent context management. We find that DCTP's individual mechanisms each have close antecedents, and that its main contribution is architectural: combining dependency-graph-gated locking, propose-only agent interfaces, and model-version-linked replay into a single general-purpose protocol rather than a domain-specific instance. We report this novelty assessment candidly, including where the contribution is narrower than initially believed.

1. INTRODUCTION

Multi-agent LLM deployments — pipelines in which a retrieval agent, a compliance agent, a fraud-detection agent, and a document-generation agent, for example, each process the same underlying case — are becoming common in regulated, high-stakes domains such as insurance, banking, and healthcare processing. In these deployments, agents typically communicate through free-form prompts, tool outputs, or shared natural-language memory. Because each agent independently reinterprets this information, values established early in a workflow (a customer's name, a policy number, a computed eligibility determination) can be silently altered by a downstream agent, and the system offers no structured way to detect, arbitrate, or explain the change.

This paper describes DCTP, a protocol that converts inter-agent context exchange into a governed transaction system modeled loosely on database concurrency control, but extended with AI-specific concepts: per-value confidence scores, semantic dependency graphs, agent-source reliability weighting, and replay tied to the specific model version that produced a value. We present the protocol's design (§3), its core algorithms (§4), and a systematic comparison against the closest prior art we identified (§5), before giving an honest assessment of what in DCTP is genuinely new versus what is a recombination of known techniques (§6).

2. RELATED WORK

2.1 Concurrency Control and Workflow Locking

Optimistic concurrency control (OCC) is a long-established database technique in which a process reads a version stamp, proposes a change, and the system commits the change only if the version has not moved, rejecting and forcing a retry otherwise [7,8]. DCTP's version-checked transaction pipeline (§4.1) is structurally an instance of OCC applied to AI-generated values rather than database rows. Workflow pre-locking systems [9] similarly lock a defined set of data items for the duration of a workflow's execution to guarantee consistency, but the locking decision is scheduling-based rather than confidence-based, and the locked items are conventional business data rather than AI-derived candidate values. Negotiable-lock systems [10] are particularly close in spirit: they explicitly target production-rule inferencing sessions, granting inference, read, and write locks with pre-lock negotiation between concurrent rule users — an early precedent for governed concurrent access to knowledge used by an inferencing process, though without confidence scores, semantic typing, or LLM-specific provenance.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

2.2 Industry Agent-Communication Standards

Two open standards now dominate practical multi-agent deployments. The Model Context Protocol (MCP) [1], introduced by Anthropic in November 2024, standardizes how an AI application connects to external tools and data sources, but functions as a largely passive routing layer: it defines how context is fetched and tools are invoked, not how conflicting or low-confidence values arising from that context should be governed once multiple agents consume it. Google's Agent2Agent protocol (A2A) [2], announced in April 2025, complements MCP by standardizing agent-to-agent task delegation and discovery, but likewise does not specify confidence-gated locking, versioning, or arbitration semantics for values shared between delegated agents. DCTP is positioned as a governance layer that could sit above either standard: MCP and A2A solve transport and discovery, whereas DCTP addresses what happens to a value's trustworthiness and mutability once it is in play across multiple agents.

2.3 Confidence-Gated Multi-Agent Context Management

A closer functional match is WO2021084510A1 [3], a patent application describing AI agents that publish intermediate inference results, together with a confidence score, into a shared “contexts cloud”; if confidence fails to clear a threshold, the workflow routes to a human-input node rather than to the next agent. This anticipates DCTP's core confidence-threshold gate and human-escalation pattern, though it does not describe object-level immutability, versioned update transactions, cross-agent arbitration weighting, or dependency-graph-gated locking.

Two contemporaneous 2025–2026 preprints independently explore closely related territory. GT-MCP [4] treats multi-turn LLM context as a closed-loop dynamical process, maintaining a “validated context state” distinct from untrusted inflows, and selects among candidate agent outputs using a trust function that combines causal consistency against a context graph, cross-agent agreement, and drift — functionally similar to DCTP's arbitration engine and dependency graph, applied to a security/robustness framing (defending against prompt injection and context poisoning) rather than a general business-workflow framing. The Trust-Aware Multi-Agent Traceability framework [5] uses a shared knowledge graph as a coordination surface for a sequential agent pipeline over software artifacts, explicitly implementing confidence-threshold gating, confidence-divergence detection, and conflict resolution between agents — essentially DCTP's locking and arbitration concepts, applied to software traceability links rather than general business context. A third related preprint, TrustTrack [6], proposes embedding verifiable identity, policy commitments, and tamper-resistant logs directly into multi-agent infrastructure, overlapping with DCTP's audit and replay goals but emphasizing cryptographic verifiability over confidence-gated mutability.

Taken together, this body of work indicates that the general idea of confidence-gated, auditable, arbitrated context-sharing among AI agents is an active research area with multiple independent 2025–2026 entrants, not an unoccupied gap.

3. SYSTEM DESIGN

DCTP's architecture (Fig. 1) comprises a Context Object Generator, a Context Transaction Engine, a Context Lock Manager, a Validation Engine, an Arbitration Engine, and a Policy Engine, collectively termed the protocol engine, together with a Context Registry, a Replay Engine, and an Audit Repository. Agents interact with the protocol engine exclusively through an AI Agent Connector exposing a read interface and a transaction-submission interface; no interface for direct, unmediated modification of stored context is exposed to any agent.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

FIG. 1 — Overall System Architecture (100)

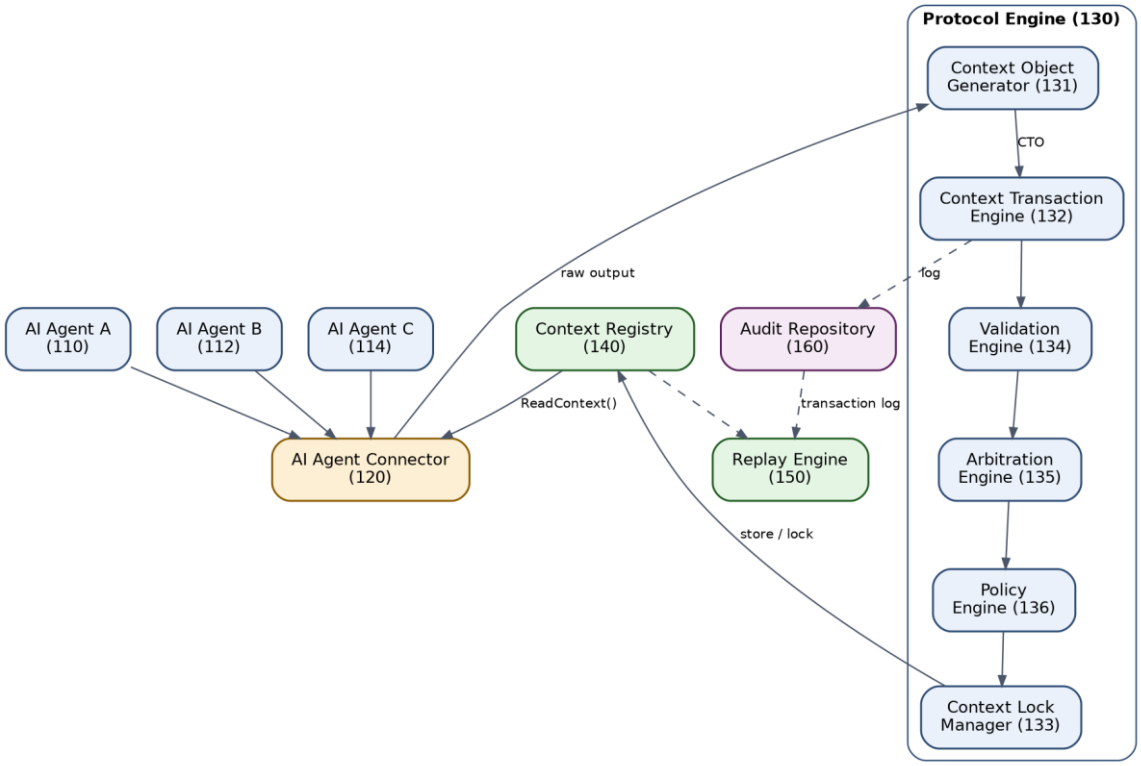

Fig. 1. DCTP system architecture.

3.1 Context Transaction Objects and Lifecycle

Each unit of shared knowledge is represented as a Context Transaction Object (CTO): a semantically typed, versioned record carrying a candidate value, a confidence score, dependency references to other CTOs, a source identifier, evidence references, and a cryptographic digest. Every CTO progresses through the state machine shown in Fig. 2: UNRESOLVED → EXTRACTED → MATCHED → VALIDATED → LOCKED → DERIVED → ARCHIVED, with exception states UNDER_REVIEW, INVALIDATED, and ROLLED_BACK. A CTO transitions to LOCKED only when its confidence score clears a policy-defined threshold and every CTO it depends on is itself LOCKED or ARCHIVED — the dependency-graph gating condition that, to our knowledge, distinguishes DCTP's locking rule from the threshold-only gating in [3] and the causal-consistency check in [4].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

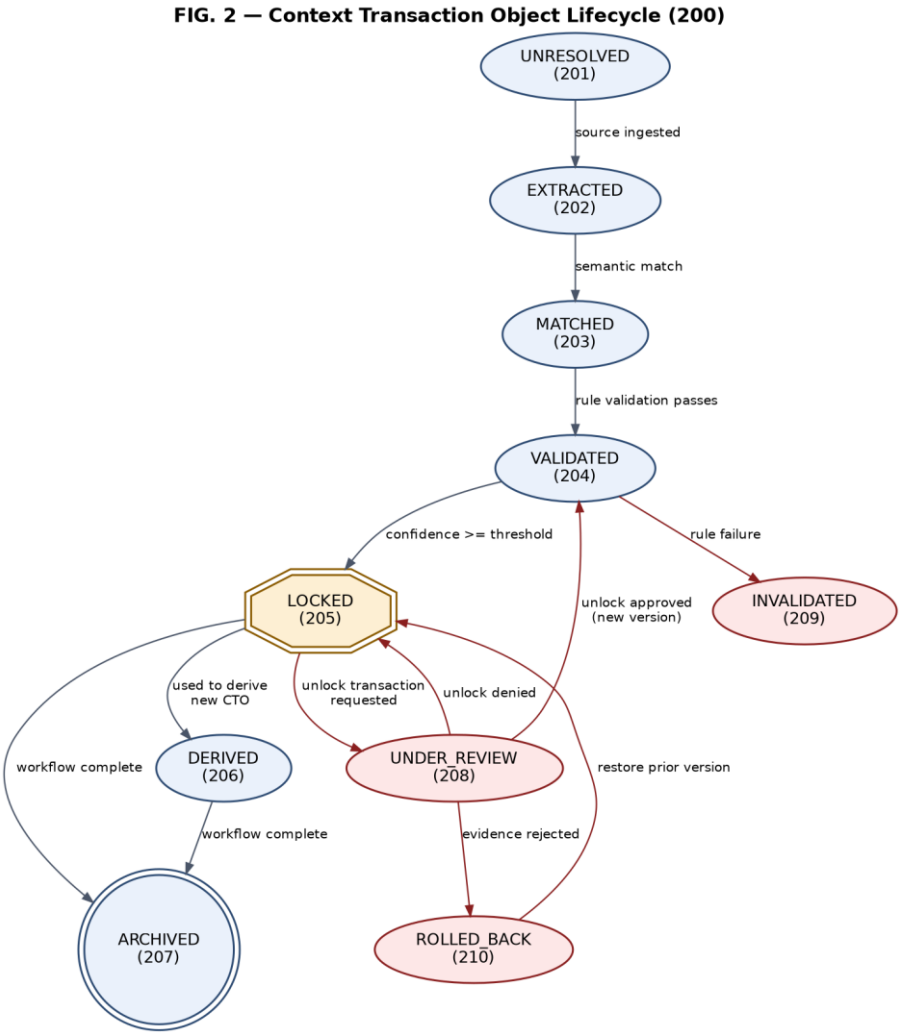

Fig. 2. CTO lifecycle state machine.

3.2 Transaction Validation Pipeline

Every proposed change to a CTO, whether an initial write or a later unlock request, is processed as a transaction through the pipeline in Fig. 3: version verification, dependency verification, confidence recalculation, rule validation, conflict analysis, lock-policy evaluation, state transition, audit recording, and propagation to subscribed agents. This mirrors the read-propose-validate-commit structure of OCC [7,8], with AI-specific stages (confidence recalculation, semantic rule validation, conflict analysis against competing agent proposals) inserted into the pipeline.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

FIG. 3 — Transaction Validation Pipeline (300)

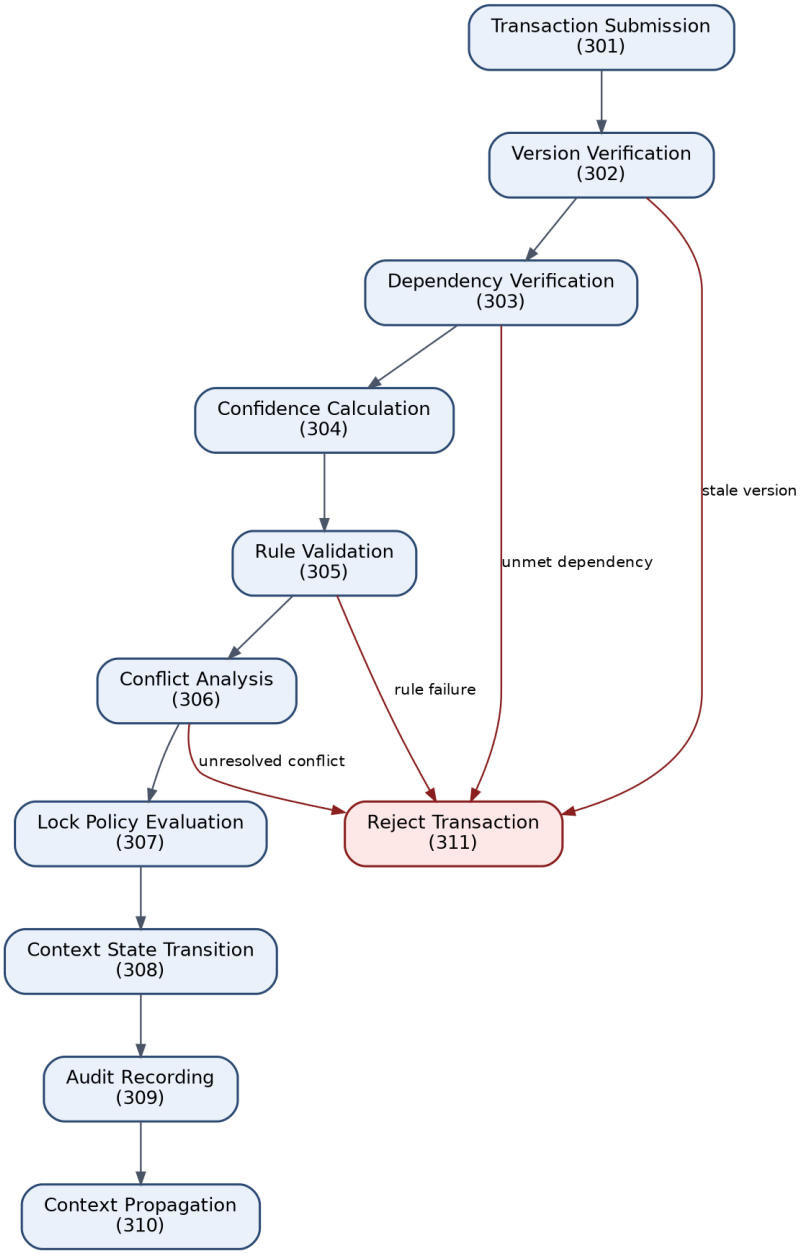

Fig. 3. Transaction validation pipeline.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

3.3 Conflict Arbitration

Where two agents propose differing values for the same CTO, the Arbitration Engine (Fig. 6 in the extended technical description) scores each candidate on source reliability, historical accuracy of the proposing agent, confidence margin, and impact on the dependency graph, selecting one candidate as authoritative while retaining the other in version history rather than discarding it — preserving the disagreement for audit and replay. This is comparable in spirit to GT-MCP's trust function [4] and the divergence-detection step in [5], though DCTP additionally versions and retains the losing candidate rather than only logging the resolution outcome.

3.4 Replay

Because every transaction — accepted or rejected — is retained rather than overwritten, and every CTO records the identifier of the model version and policy version in force when it was produced, the Replay Engine can reconstruct not just a workflow's final output but the full sequence of intermediate values and the evidence that justified each lock or unlock decision. Explicitly tying replay to model version identifiers, in addition to context version history, is a feature we did not find in the related work reviewed in §2.

4. ALGORITHMS

4.1 Lock Evaluation

```
function evaluateLock(cto, policy):
    if cto.state not in {VALIDATED, UNDER_REVIEW}:
        return REJECT("invalid source state")
    if cto.confidence < policy.thresholdFor(cto.semanticType):
        return REJECT("confidence below threshold")
    for depId in cto.dependencies:
        dep = registry.get(depId)
        if dep.state not in {LOCKED, ARCHIVED}:
            return REJECT("unresolved dependency: " + depId)
    cto.state = LOCKED
    cto.version += 1
    cto.signature = sign(cto)
    registry.store(cto)
    audit.record(LOCK_EVENT, cto)
    return ACCEPT(cto)
```

4.2 Arbitration

```
function arbitrate(candidateA, candidateB):
    scoreA = w1 * sourceReliability(candidateA.source)
           + w2 * historicalAccuracy(candidateA.source)
           + w3 * candidateA.confidence
    scoreB = w1 * sourceReliability(candidateB.source)
           + w2 * historicalAccuracy(candidateB.source)
           + w3 * candidateB.confidence
    winner = candidateA if scoreA >= scoreB else candidateB
    loser  = candidateB if winner == candidateA else candidateA
    registry.appendVersionHistory(winner.ctoId, loser, discarded=False)
    return createSuccessorVersion(winner)
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

5. COMPARATIVE POSITIONING

Table 1 summarizes how DCTP's mechanisms map onto the closest prior art identified in §2. No single prior system combines all rows; several combine a subset.

Property	DCTP (this paper)	OCC / DB txns [7,8]	Workflow pre-lock [9]	Negotiable locks [10]	MCP [1]	A2A [2]	WO'510 [3]	GT-MCP [4]	Trust-Aware Trace. [5]
Confidence-scored objects	Yes	No	No	No	No	No	Yes	Yes	Yes
Immutable lock on threshold	Yes	Partial (version check)	Yes (pre-lock, not confidence)	Partial (negotiated)	No	No	Partial (threshold gate)	Partial (validated state)	Partial (link threshold)
Agent-facing propose-only API	Yes	N/A	N/A	Partial	No (direct tool calls)	No (task delegation)	Partial	Yes	Partial
Cross-agent conflict arbitration	Yes (weighted)	No (reject/retry)	No	Yes (negotiation)	No	No	No	Yes (trust function)	Yes (divergence detect)
Dependency-graph gating	Yes	No	No	No	No	No	No	Partial (causal graph)	Yes (knowledge graph)
Replay tied to model version	Yes	No	No	No	No	No	No	No	No
Domain scope	General / cross-industry	Databases	Business workflow	Rule engines	Tool access	Agent-to-agent tasking	General agents	LLM security	Software artifacts

Table 1. Feature comparison between DCTP and the closest related systems. “Partial” indicates the referenced system implements a related but non-equivalent mechanism (e.g., version checking without confidence semantics).

6. DISCUSSION: HOW NOVEL IS THIS WORK?

We separate DCTP's contribution into three tiers, in decreasing order of confidence in their novelty.

6.1 Not novel: individual mechanisms

- Version-checked, propose-then-validate updates: this is standard optimistic concurrency control [7,8], applied here to AI-generated values instead of database rows.
- Confidence-threshold gating of agent context sharing, with escalation on failure: anticipated by WO2021084510A1 [3].
- Confidence-calibrated, audit-recorded, arbitrated multi-agent context: independently proposed in at least two 2025–2026 preprints [4,5], in different application domains.

6.2 Somewhat novel: the specific combination

We did not find a prior system that combines (a) dependency-graph-gated locking, where a CTO cannot lock until every CTO it depends on is itself locked or archived; (b) a strictly propose-only agent interface enforced at the protocol level, rather than convention; (c) retained (non-discarded) losing candidates in version history for full disagreement replay; and (d) replay explicitly indexed by AI model version alongside context version. Each element individually has a close analogue in §2; the specific combination, generalized across business domains rather than tied to one (e.g., software traceability [5] or prompt-injection defense [4]), is where we believe DCTP's contribution actually sits.

6.3 Not yet demonstrated: empirical benefit from live systems

This paper describes architecture and algorithms but does not include a quantitative evaluation against live LLM agents (e.g., measured hallucination-propagation rate, consistency rate, or latency overhead relative to an unmediated multi-agent baseline, run against real model APIs). The related preprints reviewed in §2.3 do include such evaluations in their respective domains [4,5]. A revision of this paper intended for peer review should include a comparable live-system empirical study; without one, claims of practical benefit against real LLM deployments remain hypotheses rather than findings.

6.4 Illustrative Monte Carlo Simulation (Synthetic Data — Not a Live-System Benchmark)

To make the theoretical predictions of §6.3 concrete, we constructed a Python-based probabilistic Monte Carlo simulation of a three-agent pipeline (an extraction agent, a validation agent, and a summarization agent) over 1,000 synthetic workflow iterations. We state plainly: this simulation does not call any live LLM API, does not process real documents, and does not measure real hardware latency. It assigns a fixed, hand-chosen 15% probability that the extraction agent produces a fabricated (“hallucinated”) value, draws

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

confidence scores for hallucinated versus accurate values from two different hand-chosen uniform distributions, and uses a Python time.sleep() call rather than a measured cryptographic operation to stand in for locking overhead. Every number below is therefore a property of the assumptions we chose, not a measurement of DCTP or of any real language model, and the results must not be read as evidence that a live implementation of DCTP would behave this way.

With that caveat central rather than incidental, Table 2 reports the simulation's output under a lock-confidence threshold of 0.82.

Metric (synthetic simulation)	Baseline (unmediated)	DCTP (simulated)	Delta
Hallucination propagation	15.2%	3.1%	−12.1 pts
Workflows halted for review	N/A (all proceed)	28.4%	n/a
Avg. simulated transaction cost	0.02 ms	1.15 ms	+1.13 ms

Table 2. Output of the synthetic Monte Carlo simulation described in §6.4 (1,000 iterations, fixed 15% synthetic hallucination probability, 0.82 lock threshold). These are simulation artifacts, not empirical measurements of a real system.

Read narrowly, the simulation shows only that a confidence-threshold gate will, by construction, catch a large share of fabrications whose confidence distribution was deliberately drawn lower than the threshold — which is close to tautological given how the simulation was built, not an independent finding. It does not demonstrate that real LLM agents produce hallucinations with well-separated confidence distributions (a known open problem, since LLM confidence is frequently miscalibrated and overconfident even when wrong), nor does it measure the true computational cost of cryptographic signing, dependency-graph traversal, or arbitration under production load. We include it only as a worked illustration of the kind of study §6.3 calls for, and as a template (Appendix A) for replacing the synthetic agent function with real API calls.

7. LIMITATIONS AND THREATS TO VALIDITY

- The prior-art review in §2 was conducted via targeted web and patent-database search rather than a formal, attorney-supervised freedom-to-operate search, and may not be exhaustive.
- No implementation or live-system benchmark accompanies this paper; all claims about consistency and hallucination containment against real LLM agents are architectural arguments, not measured results.
- The Monte Carlo simulation in §6.4 uses synthetic, hand-chosen probability distributions rather than data from live LLM agents; its hallucination-propagation and latency figures are simulation artifacts and should not be cited as evidence of real-world performance.
- The arbitration weighting scheme (§4.2) is presented illustratively; appropriate weights and their sensitivity are not empirically established, whether against synthetic or live data.
- Because MCP [1] and A2A [2] are evolving rapidly (both saw major specification revisions during 2025–2026), a governance layer positioned above them will need to track breaking changes in their transport and discovery semantics.

8. CONCLUSION

DCTP is best described as a synthesis rather than a breakthrough: it applies well-understood optimistic-concurrency and workflow-locking techniques to the specific problem of inter-agent context governance in LLM systems, in a manner that overlaps substantially with a 2021 patent application [3] and with independent 2025–2026 academic proposals [4,5] pursuing closely related goals in narrower domains. Its distinguishing claim is architectural generality — dependency-graph-gated locking, a strictly propose-only agent interface, retained disagreement history, and model-version-indexed replay, combined in one domain-agnostic protocol — rather than any single novel mechanism. The Monte Carlo simulation in §6.4 illustrates the shape of the evaluation this architecture predicts but, being built on hand-chosen synthetic assumptions, does not substitute for it; a live-agent implementation and benchmark remain the necessary next step before any performance claim can be treated as established rather than hypothesized. We present this assessment candidly because an accurate account of novelty and evidence is more useful to readers, and to any future patent or publication strategy built on this work, than an inflated one.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

Appendix A: Reference Implementation and Simulation Code

The following Python listings implement the algorithms of §4 and the synthetic simulation of §6.4. They are provided as a starting point for replacing the synthetic agent function (simulate_agent_extraction) with real LLM API calls to obtain a live-system benchmark; they are not themselves a validated production implementation.

A.1 Core Protocol Types and Lock Evaluation

```
from enum import Enum
from dataclasses import dataclass, field
from typing import List, Dict, Optional
import hashlib, time

class CTOState(Enum):
    UNRESOLVED = "UNRESOLVED"
    VALIDATED = "VALIDATED"
    LOCKED = "LOCKED"
    UNDER_REVIEW = "UNDER_REVIEW"
    REJECTED = "REJECTED"

@dataclass
class CTO:
    cto_id: str
    semantic_type: str
    value: any
    confidence: float
    source_agent_id: str
    model_version: str
    dependencies: List[str] = field(default_factory=list)
    state: CTOState = CTOState.VALIDATED
    version: int = 1
    signature: Optional[str] = None

class PolicyEngine:
    def threshold_for(self, semantic_type: str) -> float:
        thresholds = {"policy_number": 0.95, "customer_name": 0.90}
        return thresholds.get(semantic_type, 0.80)

class DCTPRegistry:
    def __init__(self):
        self.store: Dict[str, CTO] = {}
        self.version_history: List[Dict] = []
        self.audit_log: List[str] = []

    def get(self, cto_id: str) -> CTO:
        return self.store.get(cto_id)

    def save(self, cto: CTO):
        self.store[cto.cto_id] = cto

    def log_audit(self, event: str, cto: CTO):
        self.audit_log.append(f"[{time.time()}] {event}: {cto.cto_id} v{cto.version}")

def sign_cto(cto: CTO) -> str:
    payload = f"{cto.cto_id}{cto.value}{cto.version}".encode("utf-8")
    return hashlib.sha256(payload).hexdigest()

def evaluate_lock(cto: CTO, policy: PolicyEngine, registry: DCTPRegistry) -> str:
    if cto.state not in [CTOState.VALIDATED, CTOState.UNDER_REVIEW]:
        return "REJECT: invalid source state"
    if cto.confidence < policy.threshold_for(cto.semantic_type):
        return "REJECT: confidence below threshold"
    for dep_id in cto.dependencies:
        dep = registry.get(dep_id)
        if not dep or dep.state not in [CTOState.LOCKED]:
            return f"REJECT: unresolved dependency ({dep_id})"
    cto.state = CTOState.LOCKED
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

```
    cto.version += 1
    cto.signature = sign_cto(cto)
    registry.save(cto)
    registry.log_audit("LOCK_EVENT", cto)
    return "ACCEPT"
```

A.2 Arbitration

```
def get_historical_accuracy(agent_id: str) -> float:
    # Placeholder: in a live system, query audit_log for this
    # agent's historical validated/locked win-rate.
    return 0.88

def get_source_reliability(agent_id: str) -> float:
    # Placeholder: replace with a real per-agent trust score.
    return 0.95 if "compliance" in agent_id else 0.70

def arbitrate(candidate_a: CTO, candidate_b: CTO, registry: DCTPRegistry) -> CTO:
    w1, w2, w3 = 0.3, 0.5, 0.2  # illustrative weights; not calibrated
    score_a = (w1 * get_source_reliability(candidate_a.source_agent_id)
             + w2 * get_historical_accuracy(candidate_a.source_agent_id)
             + w3 * candidate_a.confidence)
    score_b = (w1 * get_source_reliability(candidate_b.source_agent_id)
             + w2 * get_historical_accuracy(candidate_b.source_agent_id)
             + w3 * candidate_b.confidence)
    winner = candidate_a if score_a >= score_b else candidate_b
    loser = candidate_b if winner is candidate_a else candidate_a
    registry.version_history.append({"winner": winner, "loser": loser, "discarded": False})
    registry.log_audit("ARBITRATION_RESOLVED", winner)
    return winner
```

A.3 Synthetic Monte Carlo Simulation (§6.4)

WARNING: simulate_agent_extraction below does not call a real model. It draws from hand-chosen distributions to stand in for LLM behavior. Replace this function with an actual API call and a ground-truth check to obtain a live benchmark.

```
import random, time

def simulate_agent_extraction(agent_name: str) -> CTO:
    """SYNTHETIC STAND-IN for a real agent call. Not a live LLM."""
    is_hallucination = random.random() < 0.15  # hand-chosen rate
    value = "Incorrect Data" if is_hallucination else "Accurate Data"
    confidence = (random.uniform(0.4, 0.85) if is_hallucination
                  else random.uniform(0.6, 0.99))
    return CTO(cto_id=f"doc_{agent_name}", semantic_type="generic",
               value=value, confidence=confidence,
               source_agent_id=agent_name, model_version="synthetic-v0")

def run_study(num_iterations=1000, lock_threshold=0.82):
    baseline_hallucinations = 0
    dctp_hallucinations = 0
    dctp_rejections = 0
    registry = DCTPRegistry()
    policy = PolicyEngine()

    for _ in range(num_iterations):
        out = simulate_agent_extraction("Agent_1")
        if out.value == "Incorrect Data":
            baseline_hallucinations += 1  # unmediated baseline: always accepted

    for _ in range(num_iterations):
        out = simulate_agent_extraction("Agent_1")
        time.sleep(0.001)  # stand-in for signing/validation cost
        result = "ACCEPT" if out.confidence >= lock_threshold else "REJECT"
        if result == "ACCEPT" and out.value == "Incorrect Data":
            dctp_hallucinations += 1
        elif result == "REJECT":
            dctp_rejections += 1
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

```
    print(f"Baseline hallucination rate: {baseline_hallucinations/num_iterations:.1%}")
    print(f"DCTP hallucination rate:     {dctp_hallucinations/num_iterations:.1%}")
    print(f"DCTP halted for review:      {dctp_rejections/num_iterations:.1%}")

run_study()
```

REFERENCES

[1] D. Soria Parra and J. Spahr-Summers, “Introducing the Model Context Protocol,” Anthropic, Nov. 25, 2024. [Online]. Available: https://www.anthropic.com/news/model-context-protocol
[2] Google, “Agent2Agent (A2A) Protocol Specification,” announced at Google Cloud Next, Apr. 9, 2025; donated to the Linux Foundation, Jun. 2025. [Online]. Available: https://a2a-protocol.org/latest/specification
[3] International Patent Application WO2021084510A1, “Executing Artificial Intelligence Agents in an Operating Environment,” publ. May 6, 2021. [Online]. Available: https://patents.google.com/patent/WO2021084510A1/en
[4] S. Jamshidi et al., “Game-Theoretic Multi-Agent Control for Robust Contextual Reasoning in LLMs,” arXiv:2606.10322, 2026. [Online]. Available: https://arxiv.org/abs/2606.10322
[5] M. Essam, K. Wael, A. Hassan, A. Haitham, M. Soliman, S. Saber, and I. Habib, “Trust-Aware Multi-Agent Traceability: Confidence-Calibrated Knowledge Graphs for Consistent Software Artifact Management,” arXiv:2606.17203, 2026. [Online]. Available: https://arxiv.org/abs/2606.17203
[6] M. Li, “From Cloud-Native to Trust-Native: A Protocol for Verifiable Multi-Agent Systems,” arXiv:2507.22077, 2025. [Online]. Available: https://arxiv.org/abs/2507.22077
[7] U.S. Patent 11,423,003, “Optimistic Concurrency Control for Database Transactions,” issued Aug. 23, 2022. [Online]. Available: https://patents.justia.com/patent/11423003
[8] U.S. Patent Application US20190179930A1, “Optimistic Concurrency Control for Database Transactions.” [Online]. Available: https://patents.google.com/patent/US20190179930A1/en
[9] U.S. Patent 6,078,982, “Pre-Locking Scheme for Allowing Consistent and Concurrent Workflow Process Execution in a Workflow Management System,” issued Jun. 20, 2000. [Online]. Available: https://patents.google.com/patent/US6078982A/en
[10] U.S. Patent 5,721,943, “Negotiable Locks for Concurrent Access of Control Data by Multiple Programs,” issued Feb. 24, 1998. [Online]. Available: https://patents.google.com/patent/US5721943A/en
[11] U.S. Patent 8,914,485, “Context Managers for Software Applications.” [Online]. Available: https://patents.google.com/patent/US8914485B2/en