Proof-Carrying Numbers (PCN): A Protocol for Trustworthy Numeric Answers from LLMs via Claim Verification

Anonymous authors
Paper under double-blind review

Abstract

Large Language Models (LLMs) as stochastic systems may generate numbers that deviate from available data, a failure known as *numeric hallucination*. Existing safeguards—retrieval-augmented generation, citations, and uncertainty estimation—improve transparency but cannot guarantee fidelity: fabricated or misquoted values may still be displayed as if correct. We propose **Proof-Carrying Numbers (PCN)**, a presentation-layer protocol that enforces numeric fidelity through mechanical verification. Under PCN, numeric spans are emitted as *claim-bound tokens* tied to structured claims, and a verifier checks each token under a declared policy (e.g., exact equality, rounding, aliases, or tolerance with qualifiers). Crucially, PCN places verification in the *renderer*, not the model: only claim-checked numbers are marked as verified, and all others default to unverified. This separation prevents spoofing and guarantees fail-closed behavior. We formalize PCN and prove soundness, completeness under honest tokens, fail-closed behavior, and monotonicity under policy refinement. PCN is lightweight and model-agnostic, integrates seamlessly into existing applications, and can be extended with cryptographic commitments. By enforcing verification as a mandatory step before display, PCN establishes a simple contract for numerically sensitive settings: *trust is earned only by proof*, while the absence of a mark communicates uncertainty.

1 Introduction

Large Language Models (LLMs) are emerging as powerful interfaces for accessing knowledge in domains ranging from healthcare and finance to economics and international development. Their fluency makes them attractive to a wide range of users—from policymakers and researchers to clinicians, financial analysts, and the public—but their usefulness is constrained by their stochastic nature: they may generate **numeric hallucinations**.

Even when given correct input, LLMs may still produce plausible but incorrect values—sometimes citing the right dataset while presenting the wrong figure (Ji et al., 2023; Banerjee et al., 2024; Xu et al., 2025; Kalai et al., 2025). For example, Wu et al. (2025a) showed that when provided a perturbed drug dosage, an LLM sometimes "corrected" it to a different value. In another case, a model might state that the Philippines' GDP growth in 2024 was 6% when the official figure published by The World Bank (2025) was 5.7%. Small deviations like these can erode trust and cascade into flawed medical guidance, misinformed policy, or reputational risks for institutions.

Existing safeguards only partially address this problem. Retrieval-augmented generation (Lewis et al., 2020) grounds answers in source text, while citations and attribution frameworks (Wu et al., 2025a; Zhang et al., 2025) increase transparency. However, both remain probabilistic: users often assume a cited number is faithful even when it has been misquoted or fabricated (Wu et al., 2025b; Hakim et al., 2025). Similarly, uncertainty estimation (Manakul et al., 2023) and self-verification approaches can flag suspicious values but offer no binding guarantee.

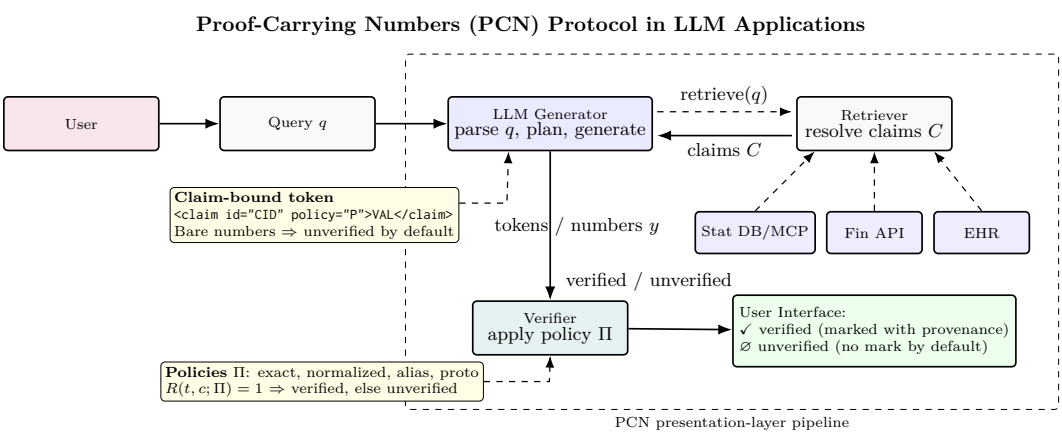

Figure 1: PCN-compliant architecture with LLM-initiated retrieval. The LLM first parses the query and *requests* claims from the retriever (dashed top lane); the retriever returns the claim set C on a separate solid lane. The LLM emits claim-bound tokens (or bare numbers). The verifier checks tokens under policy Π, and the UI renders verified values with provenance marks; absence of a mark implies unverified by default. External structured data sources feed the retriever via parallel dashed feeders.

We argue that numeric hallucination is best understood as a **presentation-layer problem**. Even when authoritative claims are retrieved, LLMs are unreliable at reproducing values faithfully (Banerjee et al., 2024; Xu et al., 2025), and user interfaces lack systematic safeguards against drift or fabrication.

To address this gap, we propose **Proof-Carrying Numbers (PCN)**, a protocol that requires every displayed number to be bound to an authoritative claim and verified before presentation. Loosely inspired by proof-carrying code (Necula, 1997), PCN embeds verifiability directly into the interface: verified numbers carry explicit provenance, while unverifiable ones are blocked, flagged, or corrected.

Our contributions are threefold:

1. We reframe numeric hallucination as a *presentation-layer problem*, showing why existing safeguards (retrieval, citations, uncertainty estimation) cannot provide binding guarantees.
2. We design the **Proof-Carrying Numbers (PCN)** protocol, specifying a claim schema, token syntax, and verifier policies that enforce a fail-closed contract at display time.
3. We show how PCN wraps inherently fallible LLM outputs in a deterministic contract: numeric spans are either *Verified* against authoritative claims with provenance, remain *Bare* if unclaimed, or are *Flagged* when verification fails.

By embedding verification into the presentation pipeline, PCN bridges the gap between LLM fluency and the trustworthiness required in high-stakes numeric applications.

2 Background and Related Work

Hallucination in large language models (LLMs)—the generation of fluent but incorrect content—poses serious challenges across domains (Ji et al., 2023; Banerjee et al., 2024; Xu et al., 2025). Of particular concern is *numeric hallucination*, where even small deviations (e.g., reporting 6.0% instead of 5.7%) can undermine high-stakes applications in policy, healthcare, and finance (Kim et al., 2025; Kang & Liu, 2023).

One stream of work grounds model outputs in retrieved content. Retrieval-augmented generation (Lewis et al., 2020) and citation frameworks (Wu et al., 2025a; Schreieder et al., 2025; Zhang et al., 2025) improve transparency by linking generated text to sources. However, fabricated values may still appear alongside credible references, creating the illusion of fidelity (Wu et al., 2025b; Hakim et al., 2025).

Another line of research focuses on post-hoc verification. Frameworks such as FEVER (Thorne et al., 2018), FEVEROUS (Aly et al., 2021), TabFact (Chen et al., 2020), and SciFact (Wadden et al., 2022) decompose outputs into claims and check them against evidence. Recent tools like AttributionBench (Li et al., 2024) and SourceCheckup (Wu et al., 2025a) extend this approach to LLMs. While useful for auditing, these methods are retrospective: they may flag erroneous outputs but cannot prevent unverified numbers from being displayed. Uncertainty-based methods (Manakul et al., 2023) similarly attempt to detect hallucinations using entropy (Farquhar et al., 2024), calibration (Manakul et al., 2023), or self-consistency (Kadavath et al., 2022), but confident models may still produce incorrect values with low uncertainty.

Structured decoding and symbolic grounding introduce additional constraints (Geng et al., 2023). Schema-constrained decoding enforces well-formed outputs, while symbolic methods such as SymGen (Hennigen et al., 2024) interleave generated text with explicit references to underlying data. These annotations reduce the burden of manual validation and make provenance more interpretable, but they stop short of guaranteeing fidelity: fabricated numbers can still appear structurally "valid" without being faithful.

At the data level, provenance frameworks like W3C Verifiable Credentials (W3C) use public key infrastructure to certify origin. While effective for data ingress, these assurances are lost once values are processed by an LLM, which may alter or fabricate outputs without detection.

In summary, prior work has improved transparency and auditing but cannot guarantee that the number ultimately displayed to the user is the one retrieved from an authoritative source. Proof-Carrying Numbers (PCN) addresses this gap by shifting from *annotation* and *detection* to *machine-enforced verification*. Under PCN, a number is marked as "verified" only if it is bound to an authoritative claim and passes deterministic checks at the presentation layer.

3 Problem Formalization

3.1 Context

Numeric hallucination is often framed as a retrieval problem, but many failures arise at the *presentation layer*: even when correct values are accessible, the number ultimately shown to the user may drift. PCN enforces a simple contract: a displayed numeric span is either Verified—because it can be mechanically matched to a structured claim under a declared policy Π—or it remains unverified (as *Bare* or *Flagged*). Verified marks thus provide positive guarantees, while their absence communicates uncertainty without suppressing content.

3.2 Setting

Consider a user query q to an AI application that integrates structured data through a database, API, or Model Context Protocol (MCP) server (Anthropic). The application resolves q into a finite claim set

$$C = \{c_1, c_2, \ldots, c_n\},$$

where each claim c has the form

$$c = \langle \text{claim\_id}, \text{indicator}, \text{entity}, \text{time}, v^*, u, m \rangle.$$

Here $v^* \in \mathbb{R}$ is the reference value, u the unit, and m metadata (e.g., dataset version). Claims may optionally be cryptographically signed, though PCN does not assume a particular trust model.

3.3 Claim-Bound Tokens and Bare Numbers

An LLM generates an output sequence $y = (y_1, \ldots, y_T)$ that may contain numeric spans. PCN requires numeric values to be emitted as *claim-bound tokens* (t):

```
<claim id="CID" policy="P">VAL</claim>
```

where `CID` links to some $c \in C$, `VAL` is the displayed number, and `P` optionally specifies a verification policy. Such tokens bind surface text to structured claims.

By contrast, a *Bare number* is emitted without a claim tag. Since it cannot be linked to any $c \in C$, it is always treated as unverified. Bare numbers may still appear in text, but never carry a verification mark.

3.4 Verification Relation and Policies

To decide whether a token t matches a claim c, PCN defines a verification relation $R(t, c; \Pi)$. Let $\hat{v}$ be the numeric payload of t normalized into the claim's unit u. Verification modes supported by Π include:

- **Exact match:** $R_{\text{exact}}(t, c) = 1 \iff \hat{v} = v^*$.
- **Rounded match:** for decimal precision d,
$$R^{(d)}_{\text{round}}(t, c) = 1 \iff \text{round}_d(\hat{v}) = \text{round}_d(v^*).$$
- **Alias equivalence:** for sanctioned scale/alias set S (e.g., $\{10^3, \text{K}, \text{thousand}\}$),
$$R_{\text{alias}}(t, c) = 1 \iff \exists s \in S : \ \hat{v} \cdot s = v^*.$$
- **Tolerance with qualifiers:** for tolerance parameters (δ, ρ) and qualifier set Q (e.g., {"about", "approximately"}),
$$R^{(\delta,\rho)}_{\text{tol}}(t, c) = 1 \iff \hat{v} \in [v^* - \max(\delta, \rho|v^*|), \ v^* + \max(\delta, \rho|v^*|)]$$
and t includes a qualifier in Q.

A policy Π specifies which relations are permitted. Formally,

$$R(t, c; \Pi) = 1 \iff \exists \text{ allowed mode in } \Pi \text{ such that it holds.}$$

If t has no claim reference or $R(t, c; \Pi) = 0$, the number is treated as unverified.

3.5 Running Example

Suppose C contains

$$c = \langle \text{"clm\_7ef6"}, \text{GDP growth}, \text{PHL}, 2024, 5.7, \%, m \rangle.$$

If the LLM emits

```
<claim id="clm_7ef6" policy="round1">5.7</claim>
```

and Π allows rounding to one decimal, verification succeeds since $\text{round}_1(5.7) = 5.7$. If the LLM emits

```
<claim id="clm_7ef6" policy="int">6</claim>
```

and Π allows rounding to the nearest integer, verification again succeeds since $\text{round}_0(5.7) = 6$. By contrast, if the LLM emits `6.0` or even `5.7` without a claim tag, the number is *Bare* and displayed without a verification mark.

3.6 Problem Statement

Given a query q, a claim set C, and an output sequence y with numeric spans $\{t_j\}$, the system must ensure

$\forall t_j \in y, \quad t_j$ is either verified against some $c \in C$ under policy Π, or surfaced as unverified.

The objective is to close the **presentation-layer verification gap**: every displayed number is either verifiably linked to a claim under Π or left unverified by default.

4 Proposed Approach: Proof-Carrying Numbers

We introduce **Proof-Carrying Numbers (PCN)**, a protocol that enforces numeric fidelity by requiring that values shown to users carry verifiable links to structured claims. Building on the formalization in Section 3, PCN is not a decoding constraint but a *presentation-layer contract*: every displayed number is either mechanically verified against a claim or presented as unverified. This section describes PCN's architecture, verification policies, user contract, and possible extensions.

4.1 Conceptual Overview

PCN integrates verification into the rendering pipeline. Numeric spans generated by an LLM are annotated with claim references, checked against structured data, and rendered with explicit status indicators. This design closes the fidelity gap: LLMs can generate fluent text, but only numbers that pass verification are displayed with verified badges, while all others remain Bare or Flagged.

4.2 System Architecture

As illustrated in Figure 1, PCN consists of four lightweight components:

1. **Retriever:** resolves a query q into a set of structured claims $C = \{c_1, \ldots, c_n\}$ from a data source such as a statistical database, financial API, medical record service, or MCP server.
2. **Generator:** produces an output sequence y that may include claim-bound tokens (Section 3.3). Bare numbers may also appear, but they carry no proof.
3. **Verifier:** checks each token against the claim set under policy Π, succeeding if $R(t, c; \Pi) = 1$ and otherwise labeling the value as Bare or Flagged.
4. **User Interface:** renders Verified numbers with explicit provenance marks (e.g., a badge and hoverable metadata). Bare numbers appear without a mark, while Flagged values are shown with a warning indicator. The absence of a mark *by default* communicates that a number is not guaranteed.

This architecture is modular and lightweight, making PCN applicable to any system that integrates LLMs with structured data, regardless of retrieval protocol or model choice.

4.3 Verification Policies

Applications require different levels of strictness. PCN supports a range of policies, as defined in Section 3.4, including exact equality, rounding to specified decimal places, alias equivalence (e.g., "K" for thousands), and tolerance with qualifiers (e.g., "about," "roughly"). Policies encode an explicit trade-off: stricter rules provide higher trust but lower coverage, while permissive ones expand coverage at the cost of precision. This explicit policy layer distinguishes PCN from schema-based decoding, which constrains format but not correctness.

4.4 User Contract

PCN enforces a *fail-closed* contract. Users can rely on two guarantees:

1. Values marked as Verified have been mechanically checked against a claim under policy Π and are displayed with provenance.
2. Values without such a mark are not verified, whether Bare (unclaimed) or Flagged (failed verification), and should be interpreted with caution.

This shifts the default assumption: current applications implicitly present all numbers as trustworthy, while PCN makes trust explicit and earned. This subtle change in user

experience is critical: it enables end-users—whether policymakers, clinicians, or financial analysts—to rely on verified numbers, distinguishing them from potential hallucinations.

4.5 Extensions: Cryptographic Proofs

PCN can be extended to settings requiring stronger provenance. Claims may embed cryptographic commitments such as Merkle proofs for large tables or PKI signatures for multi-provider trust chains. In such cases, verification not only checks numeric fidelity but also validates claim authenticity. These extensions strengthen tamper-evidence without altering the core contract: a number is verified only if it is mechanically tied to an authoritative claim.

5 Correctness Guarantees

We analyze the guarantees provided by PCN, given a generated sequence y, structured claim set C, acceptance policy Π, and verification relation $R(t, c; \Pi) \in \{0, 1\}$. The acceptance function is defined as:

$$A(y, C; \Pi) \mapsto \{(t_j, \text{label})\}_j$$

which labels each numeric span $t_j \in y$ as either Verified or Unverified.

5.1 Core Properties

Theorem 5.1 (Soundness)**.** *If $A(y, C; \Pi)$ labels t as Verified, then there exists a claim $c \in C$ such that $R(t, c; \Pi) = 1$.*

Proof sketch. By construction, the verifier only assigns Verified if it finds such a claim. Hence no fabricated value can be marked as Verified. ■

Theorem 5.2 (Completeness under honest tokens)**.** *If the generator emits a claim-bound token t referencing some $c \in C$ and $R(t, c; \Pi) = 1$, then A labels t as Verified.*

Proof sketch. Determinism of the verifier ensures all policy-compliant tokens are accepted. ■

Theorem 5.3 (Fail-Closed)**.** *Any span that (i) lacks a valid claim reference, (ii) references a non-existent claim, or (iii) fails verification under Π is labeled Unverified.*

Proof sketch. The acceptance function defaults to Unverified unless an explicit match is found. ■

Lemma 5.4 (Monotonicity under policy refinement)**.** *If $\Pi_1 \preceq \Pi_2$ (i.e., Π_1 is stricter), then:*

$$\{t : \textit{Verified}_{\Pi_1}(t)\} \subseteq \{t : \textit{Verified}_{\Pi_2}(t)\}.$$

Proof sketch. Tightening policies reduces coverage but never introduces false positives. ■

Implications. Applications can expose multiple presets (e.g., *strict*, *rounded*, *approximate*) with predictable effects on the Verified set. Tightening a policy cannot introduce false positives; relaxing a policy cannot demote a previously verified token.

5.2 Robustness to Spoofing

Theorem 5.5 (Renderer robustness)**.** *If verification status is computed by the renderer rather than text tokens, then adversarial attempts to inject symbols (✓, "verified", HTML tags) into y cannot cause A to mislabel an Unverified token as Verified.*

Proof sketch. Verified status is derived solely from $R(t, c; \Pi)$. Spoofed tokens are ignored by the parser and remain Unverified. ■

This property ensures that the mark itself is trustworthy and cannot be faked by prompt injection or adversarial text formatting.

5.3 Efficiency

Proposition 5.6 (Linear-time verification)**.** *Let $n \leq |y|$ be the number of numeric spans and $m = |C|$ the size of the claim set. If claims are indexed by identifier, then PCN verification runs in $O(n)$ time.*

Proof sketch. Each span lookup reduces to a hash-table access in $O(1)$. Policy checks are constant-time (rounding, alias lookup, tolerance check). ■

This ensures PCN verification remains negligible compared to LLM generation latency.

5.4 Cryptographic Tamper-Evidence (Extension)

Theorem 5.7 (Unforgeability of provenance)**.** *Assuming EUF-CMA security of the signature scheme and collision resistance of the hash, no adversary can cause A to label a tampered claim as* Verified *except with negligible probability.*

Proof sketch. Verification requires a valid signature or Merkle proof. Forging this reduces to breaking standard cryptographic assumptions. ■

5.5 Summary

Together, these results show that PCN provides:

- **Correctness:** Verified numbers always correspond to claims (5.1–5.4).
- **Robustness:** Verification marks cannot be spoofed (5.5).
- **Efficiency:** Verification cost is negligible (5.6).
- **Security:** Tamper-evidence is cryptographically guaranteed (5.7).

Unlike heuristic methods, PCN requires no probabilistic confidence scoring. Fidelity follows deterministically from the protocol's construction.

6 Discussion and Limitations

Proof-Carrying Numbers (PCN) reframes numeric hallucination not as a question of whether the model "knows" the right value, but of what the user interface is permitted to display. By enforcing a fail-closed contract, PCN ensures that numbers marked as "verified" are mechanically tied to authoritative claims, while all others are visibly unverified. This shift moves trust from probabilistic model behavior to deterministic verification at the presentation layer.

6.1 Scope of Guarantees

PCN's guarantees are intentionally modest but powerful. It does not claim that a model's reasoning is sound or that a dataset is "true." Instead, it guarantees that displayed numbers are either (i) verifiably consistent with a claim under a policy Π, or (ii) explicitly unverified. This closes one of the most dangerous loopholes in current AI systems: the undetected inclusion of fabricated numbers in fluent responses. In practice, this enables policymakers, clinicians, or analysts to treat verification marks as binding fidelity contracts, while interpreting unmarked numbers as provisional.

6.2 Policy Design and Usability

A defining feature of PCN is its *policy layer*, which governs how closely numeric spans must match reference claims to receive verification. While applications may define their own policies, data providers can also publish canonical rules (e.g., rounding conventions, tolerances) alongside claims. This reduces the risk of arbitrary or inconsistent application-level choices. For example:

- **Clinical dosage:** providers may require exact equality, reflecting zero tolerance for deviation.
- **Macroeconomic growth:** agencies may allow one-decimal rounding to match dissemination practices.
- **Journalistic communication:** datasets may permit approximate expressions ("about," "roughly") within a bounded tolerance.

Policies strengthen provenance but raise challenges of interoperability and accountability: strict rules may reduce coverage, while inconsistent ones across sources complicate multi-provider verification. The monotonicity property (Theorem 5.4) ensures such trade-offs remain predictable.

6.3 Risks and Threats

PCN's effectiveness depends on both human factors and adversarial resilience.

Verification coverage gaps. PCN only verifies numbers that the LLM tags with claim-bound tokens. If tagging recall is poor, many numeric spans will remain *Bare*—visible but without verification marks. This can create the perception that the system is unreliable, even though it reflects limitations in the LLM's compliance rather than the verifier itself. Improving prompting, fine-tuning, or constrained decoding is therefore essential to make PCN useful in practice.

Policy misconfiguration. Overly strict policies cause excessive verification failures, while permissive ones dilute guarantees. Since policies directly shape user trust, misconfiguration can either frustrate users or undermine fidelity. Clear presets (e.g., "strict," "tolerant") mitigate this risk.

Overconfidence in scope. PCN secures numeric fidelity only. Users may mistake badges for guarantees of holistic correctness, when reasoning and non-numeric facts remain outside its scope. Scope must be communicated explicitly.

Institutional responsibility. Verification ties numbers to specific providers, strengthening provenance but also shifting accountability. Custodians may hesitate to participate if they fear liability. Adoption will require governance frameworks that distribute responsibility across model providers, developers, and institutions.

Adversarial considerations. Common attack surfaces map directly onto PCN's guarantees: fabricated values cannot be marked (fail-closed), spoofed symbols do not confer verification (renderer robustness), and tampering with claim stores can be mitigated by cryptographic commitments. Additional operational safeguards (e.g., version pinning, audit logs) further reduce risk. Identifier abuse and privacy remain implementation-level considerations.

6.4 Integration and Overhead

From an engineering standpoint, PCN is lightweight. Verification runs in $O(n)$ time over numeric spans and adds negligible latency compared to model decoding. This makes it practical as a drop-in module for RAG pipelines, LLM-based chatbots, or statistical portals.

Early experiments suggest that prompting or light fine-tuning enables models to emit claim tags with reasonable recall, though further empirical work is needed.

6.5 Broader Implications

By decoupling fluency from fidelity, PCN reshapes incentives for trust in AI systems:

- **Developers** can build applications where trust derives from the verification pipeline rather than the model itself.
- **Institutions** (e.g., central banks, ministries of health, or the World Bank) can act as trust anchors by supplying authoritative claims.
- **Users** gain a clear signal: trust is earned only by proof, and the absence of a mark is itself informative.

This reframing does not solve factuality in general, but it addresses the class of errors with the highest downstream risk: misrepresented numbers. Even small numeric drifts can cascade into reputational or policy harms; PCN offers a minimal but enforceable safeguard.

6.6 Limitations and Future Work

PCN has clear boundaries: it cannot guarantee reasoning correctness, covers only structured claims, and depends on the availability of authoritative data sources. Its effectiveness also hinges on LLM cooperation in emitting claim-bound tokens. Future work should evaluate user behavior around verification marks, refine policy design trade-offs, extend PCN to derived values (e.g., ratios, aggregates) by verifying deterministic functions over atomic claims, and extend PCN to multi-provider cryptographic trust chains. Ultimately, PCN aims to make verification—not assumption—the default habit of numeric communication in AI.

7 Conclusion

We introduced *Proof-Carrying Numbers* (PCN), a protocol that makes numeric fidelity a presentation-layer property. PCN binds displayed numbers to structured claims and verifies each span under an explicit policy Π, yielding formal guarantees of soundness, completeness under honest tokens, and fail-closed behavior in which Bare or invalid spans never appear as Verified. Unlike retrieval, citation, or schema-only approaches, PCN treats verification as a first-class, mechanical step between model outputs and the user interface. The result is a simple contract: *trust is earned only by proof*, while the absence of a mark communicates uncertainty without suppressing content.

PCN is domain-agnostic: claims may originate from statistical databases, clinical systems, financial APIs, or other structured sources. Its modular architecture (retriever, generator, verifier, UI) integrates with existing applications, and optional cryptographic commitments (signatures, Merkle proofs) strengthen provenance without altering the core contract.

The protocol's scope is intentionally bounded. PCN guarantees correspondence to a chosen source, not ultimate truth, and it currently addresses atomic numeric spans rather than derived expressions or free-text facts. However, closing the verification gap at the presentation layer offers a practical step toward trustworthy numeric communication in LLM applications.

Future work includes developing SDKs and reference implementations across application stacks, designing adaptive policies such as tolerances with guardrails, and studying how users interact with verification marks and provenance cues. PCN could also be extended to cover derived values (e.g., ratios and aggregates) through deterministic functions over atomic claims, and to multi-provider deployments secured by cryptographic trust chains. Taken together, these directions position PCN as a minimal yet extensible blueprint for deploying LLMs in numerically sensitive settings with explicit, inspectable guarantees.

References

Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. FEVEROUS: Fact Extraction and VERification Over Unstructured and Structured information. June 2021. URL https://openreview.net/forum?id=h-flVCIlstW.

Anthropic. Introducing the Model Context Protocol. URL https://www.anthropic.com/news/model-context-protocol.

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. LLMs Will Always Hallucinate, and We Need to Live With This, September 2024. URL http://arxiv.org/abs/2409.05746. arXiv:2409.05746 [stat].

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. TabFact: A Large-scale Dataset for Table-based Fact Verification, June 2020. URL http://arxiv.org/abs/1909.02164. arXiv:1909.02164 [cs].

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, June 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0. Publisher: Nature Publishing Group.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674. URL https://aclanthology.org/2023.emnlp-main.674/.

Joe B. Hakim, Jeffery L. Painter, Darmendra Ramcharran, Vijay Kara, Greg Powell, Paulina Sobczak, Chiho Sato, Andrew Bate, and Andrew Beam. The need for guardrails with large language models in pharmacovigilance and other medical safety critical settings. *Scientific Reports*, 15(1):27886, July 2025. ISSN 2045-2322. doi: 10.1038/s41598-025-09138-0. URL https://www.nature.com/articles/s41598-025-09138-0. Publisher: Nature Publishing Group.

Lucas Torroba Hennigen, Zejiang Shen, Aniruddha Nrusimha, Bernhard Gapp, David Sontag, and Yoon Kim. Towards Verifiable Text Generation with Symbolic References. August 2024. URL https://openreview.net/forum?id=fib9qidCpY.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, 55(12):248:1–248:38, March 2023. ISSN 0360-0300. doi: 10.1145/3571730. URL https://dl.acm.org/doi/10.1145/3571730.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language Models (Mostly) Know What They Know, November 2022. URL http://arxiv.org/abs/2207.05221. arXiv:2207.05221 [cs].

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate, September 2025. URL https://cdn.openai.com/pdf/d04913be-3f6f-4d2b-b283-ff432ef4aaa5/why-language-models-hallucinate.pdf.

Haoqiang Kang and Xiao-Yang Liu. Deficiency of Large Language Models in Finance: An Empirical Examination of Hallucination, November 2023. URL http://arxiv.org/abs/2311.15548. arXiv:2311.15548 [cs].

Yubin Kim, Hyewon Jeong, Shan Chen, Shuyue Stella Li, Mingyu Lu, Kumail Alhamoud, Jimin Mun, Cristina Grau, Minseok Jung, Rodrigo Gameiro, Lizhou Fan, Eugene Park, Tristan Lin, Joonsik Yoon, Wonjin Yoon, Maarten Sap, Yulia Tsvetkov, Paul Liang, Xuhai Xu, Xin Liu, Daniel McDuff, Hyeonhoon Lee, Hae Won Park, Samir Tulebaev, and Cynthia Breazeal. Medical Hallucination in Foundation Models and Their Impact on Healthcare, March 2025. URL https://www.medrxiv.org/content/10.1101/2025.02.28.25323115v1. Pages: 2025.02.28.25323115.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pp. 9459–9474, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 978-1-71382-954-6.

Yifei Li, Xiang Yue, Zeyi Liao, and Huan Sun. AttributionBench: How Hard is Automatic Attribution Evaluation? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14919–14935, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.886. URL https://aclanthology.org/2024.findings-acl.886/.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9004–9017, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.557. URL https://aclanthology.org/2023.emnlp-main.557/.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM Conferences, pp. 106–119. January 1997. ISBN 978-0-89791-853-4. doi: 10.1145/263699.263712. URL https://dl.acm.org/doi/10.1145/263699.263712.

Tobias Schreieder, Tim Schopf, and Michael Färber. Attribution, Citation, and Quotation: A Survey of Evidence-based Text Generation with Large Language Models, August 2025. URL http://arxiv.org/abs/2508.15396. arXiv:2508.15396 [cs] version: 1.

The World Bank. World Bank Open Data - World Development Indicators (WDI), 2025. URL https://data.worldbank.org.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a Large-scale Dataset for Fact Extraction and VERification. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1074. URL https://aclanthology.org/N18-1074/.

W3C. Verifiable Credential Data Integrity 1.0. URL https://www.w3.org/TR/vc-data-integrity/.

David Wadden, Kyle Lo, Bailey Kuehl, Arman Cohan, Iz Beltagy, Lucy Lu Wang, and Hannaneh Hajishirzi. SciFact-Open: Towards open-domain scientific claim verification. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 4719–4734, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.347. URL https://aclanthology.org/2022.findings-emnlp.347/.

Kevin Wu, Eric Wu, Kevin Wei, Angela Zhang, Allison Casasola, Teresa Nguyen, Sith Riantawan, Patricia Shi, Daniel Ho, and James Zou. An automated framework for assessing how well LLMs cite relevant medical references. *Nature Communications*,

16(1):3615, April 2025a. ISSN 2041-1723. doi: 10.1038/s41467-025-58551-6. URL https://www.nature.com/articles/s41467-025-58551-6. Publisher: Nature Publishing Group.

Kevin Wu, Eric Wu, and James Zou. ClashEval: Quantifying the tug-of-war between an LLM's internal prior and external evidence, February 2025b. URL http://arxiv.org/abs/2404.10198. arXiv:2404.10198 [cs].

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is Inevitable: An Innate Limitation of Large Language Models, February 2025. URL http://arxiv.org/abs/2401.11817. arXiv:2401.11817 [cs].

Jiajie Zhang, Yushi Bai, Xin Lv, Wanjun Gu, Danqing Liu, Minhao Zou, Shulin Cao, Lei Hou, Yuxiao Dong, Ling Feng, and Juanzi Li. LongCite: Enabling LLMs to Generate Fine-grained Citations in Long-Context QA. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5098–5122, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 9798891762565. doi: 10.18653/v1/2025.findings-acl.264. URL https://aclanthology.org/2025.findings-acl.264/.

Disclaimer and Disclosure of AI Use

Microsoft Co-Pilot and ChatGPT were employed to enhance the manuscript's readability and consistency. ChatGPT was also used to critique the previous versions of the manuscript for suggested improvements.

A Appendix

A.1 User Interface Implementing PCN

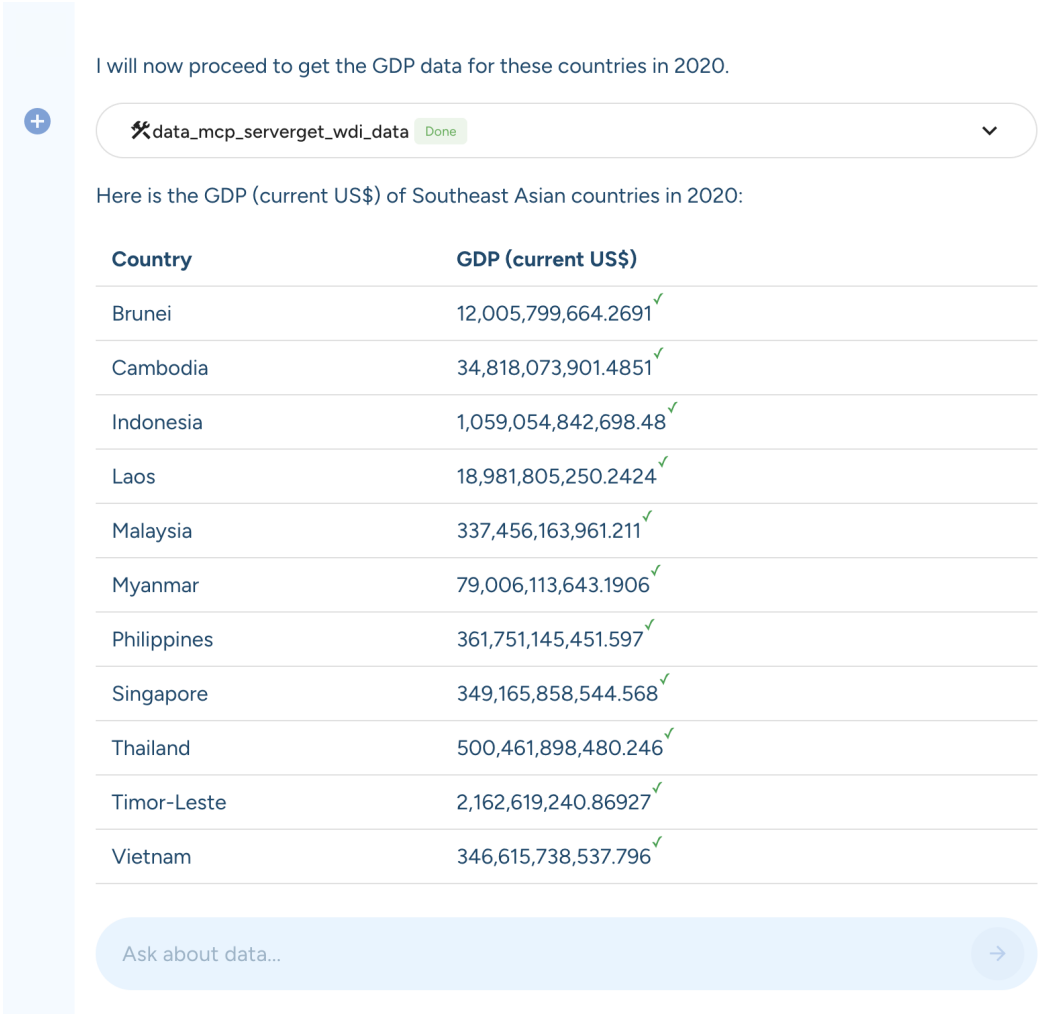

Figure 2: Implementation of the Proof-Carrying Numbers (PCN) protocol in an LLM chat application. The LLM retrieves authoritative claims via an MCP server; the verifier applies PCN policies; and the interface renders verified numbers with explicit badges.

A.2 Sample implementation

Suppose the user asks a question to the LLM: "What is the gdp growth of the Philippines in 2024?"

The following shows how the Proof-Carrying Numbers (PCN) protocol works to provide an answer to the user.

First, the LLM queries the data retriever, in this case, an MCP server that gets data from the World Development Indicators. The MCP server implements the PCN protocol and returns this PCN payload with the claim to the LLM generator.

PCN compliant claim from the retriever


```
{
  "data": [
    {
      "indicator_id": "NY.GDP.MKTP.KD.ZG",
      "indicator_name": "GDP growth (annual %)",
      "data": [
        {
          "country": "Philippines",
          "date": "2024",
          "value": 5.69201612823412,
          "claim_id": "0328"
        }
      ]
    }
  ],
  "note": {
    "NY.GDP.MKTP.KD.ZG": ""
  }
}
```


The LLM, given the payload, will generate a response that attempts to answer the user's question using the retriever's data. At the same time the LLM is instructed to conform to the PCN protocol in generating the response.

The instruction given to the LLM and its PCN-compliant response are shown below.

Example prompt to encourage PCN use by the LLM

```
[... other instructions]

When you provide any numerical data or values obtained from the tools, you must
   ↪  always enclose the numbers within a claim tag in the following format: `<claim
   ↪  id="claim_id">"value"</claim>`. For example, "The GDP of the Philippines in
   ↪  2020 is <claim id="5e1f">361,751,145,451.597</claim> USD". This is mandatory.
```

PCN-compliant LLM response

```
<p>The GDP growth rate of the Philippines in 2024 is projected to be <claim
   ↪  id="0328">5.69%</claim> (annual %).</p>
```

In parallel, when the receiver returns the PCN claim requested by the LLM, a claim aggregation process is applied to capture the claim, which can be used during verification.

The TypeScript function below shows how the PCN payload from the retriever is aggregated to create a map of claims that the verifier can use to verify the claims in the response generated by the LLM.

Extraction of claims from the retriever component

```
export function getClaims (messages: ResponseMessage[]): Record<string,
   ↪  Record<string, any>> {
  const claims: Record<string, Record<string, any>> = {}

  for (const m of messages
    .filter(m => m.role === 'tool')) {
```

```
    const parsed = m.toolContent?.output?.parsed
    if (!parsed?.data) {
      continue
    }

    for (const indicator of parsed.data) {
      for (const d of indicator.data) {
        if (d.claim_id) {
          claims[d.claim_id] = {
            country: d.country,
            date: d.date,
            value: d.value,
            indicator_id: indicator.indicator_id,
          }
        }
      }
    }
  }

  return claims
}
```

Now that the LLM has generated a response, we execute the PCN-verifier module to assess if claims have been made and then validate if any.

The TypeScript function below shows how the PCN protocol can be implemented to process the response generated by an LLM. This example implements the exact policy variant.

Processing of PCN claims in LLM content

```
const processPCNClaims = (content: string) => {
  return content.replace(
    /<claim id="([^"]+)">(.*?)<\/claim>/g,
    (match, claimId: string, innerText: string) => {
      // Remove any existing verification markers to make this idempotent
      const cleanInner = innerText.replace(
        /<sup class="(?:verified-mark|verify-pending)".*?<\/sup>/g,
        '',
      ).replace(
        /<span class="needs-verify".*?>(.*?)<\/span>/g,
        '$1',
      )

      const claim = claims.value?.[claimId]
      if (!claim) {
        // No known claim for this id → mark as pending
        return `<claim id="${claimId}">${cleanInner}<sup class="verify-pending"
          ↪  title="Needs verification" role="img" aria-label="Needs
          ↪  verification">X</sup></claim>`
      }

      // Normalize inner text & claim value (remove spaces/commas)
      const normalizedInner = String(cleanInner).replace(/[\s,]/g, '')
      const normalizedClaimValue = String(claim.value).replace(/[\s,]/g, '')

      if (normalizedInner === normalizedClaimValue) {
        // Verified
        return `<claim id="${claimId}">${cleanInner}<sup class="verified-mark"
          ↪  title="Verified data">OK</sup></claim>`
      }

      // Mismatch → needs verification
```

```
      return `<claim id="${claimId}">${cleanInner}<sup class="verify-pending"
        ↪ title="${toTitleAttr(claim.country, claim.date, String(claim.value))}"
        ↪ role="img" aria-label="Needs verification">X</sup></claim>`
    },
  )
}
```

The verifier modifies the content by injecting verification signals into the response. In this case, since the policy is exact, and the LLM opted to return a rounded-off version of the actual value received from the receiver, then the verifier returns a warning signal indicating the policy constraint was not met.

Updated response by the verifier

```
    <p>The GDP growth rate of the Philippines in 2024 is projected to be <claim
      ↪ id="0328">5.69%<sup class="verify-pending" title="Country: Philippines
Date: 2024
Value: 5.69201612823412" role="img" aria-label="Needs verification">X</sup></claim>
  ↪ (annual %).</p>
```

I found relevant indicators for GDP growth rate for the Philippines. The most relevant indicator for GDP growth (annual %) is "GDP growth (annual %)" with the indicator code NY.GDP.MKTP.KD.ZG.

Next, I will retrieve the GDP growth rate data for the Philippines for the year 2024 if available.

The GDP growth rate of the Philippines in 2024 is projected to be 5.69% ⚠ (annual %).

Figure 3: Illustration of the Proof-Carrying Numbers (PCN) protocol in an LLM chat application, where the response of the LLM (5.69%) didn't meet the exact policy set in the verification. A clear warning mark is added that hints to users to review the output.