

The deterministic gap

Generative AI can propose, explain and build at extraordinary speed. It still cannot make a consequential answer true. This paper separates inference from computation and argues for systems in which models expand the option space while explicit rules, state and evidence close it.

AUTHORED BY

Daniel Johnson & Gary McLean Hall

AI DRAFTING

Astra, Fable & the models of the moment

EVIDENCE CHECKED

15 September 2026. The date tells you when the cited material was reviewed, not when the future agreed to behave.

INVALIDATED BY

When model behaviour, evaluation evidence or governance materially changes the argument.

EXPIRY DATE

The reader's call. Before a fluent answer is mistaken for a calculation receipt.

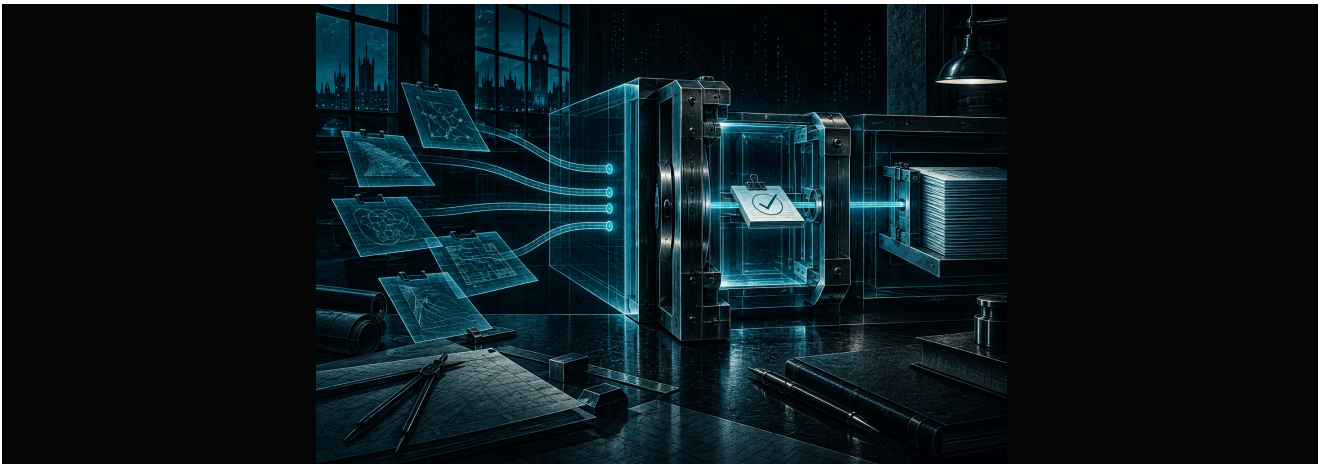
The founders own the argument; AI helped draft the prose. The sources and limits are here so you can challenge both.

Contents

- 01. The claim
 - 02. What a language model actually does
 - 03. The echo chamber of one
 - 04. Context is a workbench, not memory
 - 05. Where inference belongs
 - 06. Where determinism belongs
 - 07. The construction loop
 - 08. What the evidence does and does not support
 - 09. Design consequences
 - 10. Conclusion
 - 11. References
-

The claim

Artificial intelligence is changing the economics of software construction. It can search a design space, translate between forms, draft interfaces, propose code, explain unfamiliar systems and carry a large amount of routine implementation work. That is not the same as being a dependable system of record.



The distinction matters whenever an answer has consequences. A tax calculation, an entitlement, a balance, a statutory deadline, a permission decision or the transition of a case from one legal state to another must be reproducible and inspectable. Given the same valid state and the same applicable rule, the product must calculate the same answer. If a rule changes, the change must be visible. If somebody challenges the outcome, the organisation must be able to show how it was reached.

That is the deterministic gap: the distance between a plausible proposal and an answer an organisation is entitled to rely on.

The gap is not an argument against AI. It is the design space in which AI becomes useful without becoming sovereign. Models can generate options. Specifications, typed data, deterministic functions, permissions, tests and audit evidence decide what is allowed to survive.

The practical boundary is an act of commitment. A draft recommendation can be revised cheaply; a payment instruction, refusal of access or published statutory decision can create effects before anybody notices the error. Architecture should therefore distinguish preparation from commitment. The preparation side may contain uncertainty and competing interpretations. The commitment side needs a defined actor, validated inputs, an applicable rule and an evidence record.

This distinction also avoids demanding certainty where the domain has none. A forecast remains uncertain even when calculated by deterministic code. A professional judgement remains a judgement even when supported by excellent evidence. The system should preserve the category of the answer: observation, estimate, calculation, recommendation or decision. Problems begin when an interface presents all five with the same visual authority and the receiving organisation cannot tell which obligation it has accepted.

What a language model actually does

The [Transformer architecture](#) was introduced as a sequence-modelling system based on attention. During autoregressive text generation, a model repeatedly estimates a probability distribution over possible next tokens and samples or selects from it. The result can look like retrieval, reasoning, memory or personality because language contains traces of all of those things. That account describes a central generation mechanism, not every training objective or architecture used in current AI systems.

Inspect the arithmetic chain

Tool result

A correct sum can still use an incorrectly extracted amount.

Evidence needed

Document location → selected input → operation and rounding → result.

That mechanism is extraordinarily productive. It is also why fluency is not evidence. Pretraining rewards prediction of training text; subsequent tuning shapes instruction-following and preferred behaviour. Neither objective guarantees that a particular answer is true, and the model has no built-in warning light that reliably separates remembered fact, derived result and plausible completion.

[TruthfulQA](#) was designed to test whether models reproduce falsehoods learned from human text. Its authors reported that the best model in their tested set was truthful on 58 per cent of questions, compared with 94 per cent for human respondents, and that scaling alone did not solve the problem. Those are historical benchmark results, not estimates of today's general error rate. Their relevance here is the demonstrated distinction between producing plausible language and producing a truthful answer.

The same distinction appears in simple arithmetic. A model may emit the right total because the pattern is common, because it internally approximates a procedure, because it used a tool, or by chance. A calculator applies a defined operation. When the business depends on the total, the calculation belongs in code whose inputs, operation, rounding and output are explicit. The model can write that code. It should not impersonate it at runtime.

It would be equally mistaken to conclude that prediction cannot support reasoning. A model can construct a useful derivation, select the right tool or discover a counterexample. The issue is how the receiving system distinguishes a successful derivation from a convincing imitation. An explanation is a candidate trace until its consequential steps can be checked against sources or executed operations. A detailed answer may make that checking easier; detail alone does not perform it.

Tool use moves the boundary rather than abolishing it. Suppose an assistant calls a calculator correctly but extracts the wrong amount from a document. The arithmetic can be flawless while the business answer is wrong. A useful trace therefore records where each input came from, the transformation applied, the tool call and the result used. Verification has to cover the chain that matters to the obligation, including interpretation at its beginning and presentation at its end.

The echo chamber of one

Language models are trained on human language and then tuned with human preferences. That gives them useful social competence. It also creates a pressure to agree with the framing they are given.



[Research on sycophancy](#) found agreement-seeking behaviour in the tested assistants and showed that preference judgments could favour convincing agreement over correctness. Anthropic's [model-written evaluations](#) likewise investigated sycophancy among other behaviours. These studies do not show that every agreeable answer is false. They show that agreement is a behaviour to test, not a quality guarantee.

In practice, a conversation can become an echo chamber with one participant. The user supplies a premise. The model elaborates it, adds detail and speaks with increasing authority. The extra detail feels like corroboration even when it is only another completion

conditioned on the same premise. A request for confidence can make the prose firmer without making the evidence stronger.

This is why factual anchors must enter from outside the conversation: source documents, observed data, executable code, independent tests, registers, signed decisions and human domain judgement. Asking the same model to criticise its own unsupported answer is useful exploration, but it is not independent verification.

Independence is not merely a second model name. Two reviewers can share the same mistaken source, accept the same framing or inherit a generated summary that already removed the contradiction. Their agreement adds little if their routes to the answer are effectively identical. A stronger arrangement separates evidence acquisition, proposed interpretation and acceptance criteria. One process obtains the source; another proposes a rule; an independently specified test challenges what that rule would do.

For a contested recommendation, ask what observation would change the decision. If no answer is possible, further conversation is unlikely to improve its evidential status. Record the disputed premise, the strongest alternative interpretation and the missing fact that separates them. That gives a domain owner something concrete to resolve. It also prevents the common failure in which every objection produces a longer explanation of the original position and the final document looks better supported only because it is longer.

Context is a workbench, not memory

A model's context window is the material made available for the current inference. It behaves less like durable organisational memory and more like a workbench. Put the applicable statute, schema, architecture decision and acceptance criteria on the bench and the model can use them. Fill it with stale plans, conflicting drafts and irrelevant logs and the model must infer which instructions matter.

A stale-read check

At session start

Load the named authoritative revision.

At acceptance

Check that the shared contract has not changed beneath the work.

Lost in the Middle found that the tested models often used information less reliably when it appeared in the middle of a long context than at its beginning or end. This does not establish the same performance curve for every later model. It does establish why nominal context capacity and effective use should be measured separately on the actual task.

For engineering, the implication is concrete. The authoritative plan cannot live only in a chat. It must live in versioned artefacts: a specification, schema, decision record, test suite and operating guide. A session receives the smallest coherent slice it needs. Its output is checked against durable state before it becomes durable state itself.

A workbench needs a loading protocol. The session should know which revision is authoritative, what it may edit and what to do if a source conflicts with the current implementation. An instruction to “use the latest requirements” is insufficient when the repository contains three plausible versions. A manifest can name the exact files and their role, with decisions pointing to the source they interpret. The manifest is small because it selects authority, not because it compresses every relevant fact into a slogan.

Concurrency makes this important even when context is plentiful. Another worker may change a schema after a session reads it. Before accepting the output, the system should establish that the contract used by the worker is still the contract being released. If it changed, the affected work needs reconciliation. Longer context cannot solve a stale-read problem; versioning, ownership and a check at the acceptance boundary can.

Where inference belongs

Inference is valuable when plurality is valuable. Research questions may have several promising directions. A design problem may support many workable layouts. An unfamiliar codebase may admit several hypotheses. A human may benefit from alternative explanations before deciding.

Document intake boundary

Candidate

Extracted names and dates retain their source locations.

Committed record

Validation and authorised confirmation establish the case and its clock.

Models are therefore well suited to:

- proposing options and counterarguments;
- translating prose into candidate structures;
- drafting code from explicit contracts;
- finding inconsistencies across a bounded corpus;
- generating test cases and mutations;
- summarising evidence for a human decision; and
- operating tools within restricted permissions.

None of those uses requires treating the raw completion as a final fact. The model creates candidates; another mechanism selects, verifies or rejects them.

There is also a legitimate future for more general machine intelligence. A system able to plan across domains, use tools and revise its own approach could be transformative. Greater agency makes boundaries more important, not less. The more capable the proposer, the more consequential it becomes to define what it may change, what evidence must exist, and which decisions remain with accountable humans.

Consider an illustrative document-intake workflow. A model proposes the document type and extracts candidate dates and names. The product retains the original document and links each extracted field to its source location. Deterministic validation rejects impossible dates and unknown identifiers. Low-confidence or consequential ambiguities enter a review queue. Only an authorised confirmation creates the durable case record and starts any business clock. The model does useful work throughout, but a fluent extraction does not silently become an official fact.

This design has costs: retained provenance, review work and an explicit unresolved state. Those costs should be compared with the cost of recovering from an untraceable wrong answer. For low-consequence uses, the organisation may accept a lighter boundary. The decision should be deliberate and reversible where possible. The relevant question is what harm the output can cause before it is corrected, not whether the feature happens to contain a language model.

Where determinism belongs

Determinism is required where the organisation owes a stable answer. The relevant form is not merely “the program repeats itself.” A consistently wrong calculation is deterministic. Dependability needs three separate claims:

The payment that might already exist

A timeout leaves the external outcome unresolved. Retrying must refer to the existing instruction under a supported provider contract.

Observed	Instruction sent; acknowledgement lost.
Unknown	Whether the provider accepted the payment.
Next evidence	Query or reconcile the existing request identity.

Constructed case · a local audit row cannot prove an external payment happened once.

1. Reproducibility: the same recorded inputs and rule version produce the same output.
2. Correctness within a stated boundary: tests and evidence show that the implementation matches the agreed rule for the cases covered.
3. Legitimacy: an accountable person or authority has decided that the rule itself is appropriate.

This separation prevents a common rhetorical shortcut in AI systems. Replacing randomness with a low temperature does not establish correctness. Adding citations does not establish that the cited rule was applied. Putting a human “in the loop” does not help if the interface hides the state, rationale and alternatives required for a real decision.

A dependable product therefore records inputs, validates types and ranges, applies versioned rules, controls permissions, makes state transitions explicit and emits an audit trail. Where uncertainty is real, it represents uncertainty as data and routes it to an accountable decision rather than smoothing it into confident prose.

An illustrative deadline service makes the three claims separable. Its inputs might include a received date, a jurisdiction, a rule identifier and a versioned holiday calendar.

Reproducibility means those same recorded values recreate the same deadline.

Correctness evidence exercises the applicable start-day convention, exclusions and boundary dates. Legitimacy concerns who approved that convention and whether it is the right one for the case. A passing arithmetic test cannot settle a dispute about the governing policy.

The service should also define what happens when a required calendar is unavailable.

Guessing a likely holiday or silently using a newer calendar changes the meaning of the answer. Possible designs include refusing calculation, returning a provisional result clearly marked as such, or routing to an authorised manual decision. Which is appropriate depends on the obligation. The important point is that absence is represented as state rather than concealed by a plausible completion.

Reproduction must not rewrite history. If a rule changes next year, an old decision should remain explainable under the version actually applied. A separate reassessment may calculate the new result and record why it differs. Keeping those operations distinct protects both audit and correction: the organisation can acknowledge that an earlier rule was wrong without pretending that the earlier decision was made under a rule that did not yet exist.

A worked commitment: invoice to payment instruction

Consider a constructed invoice-processing service. A supplier sends a document containing line items, a total and bank details. A model extracts candidate fields. The document is evidence of what the supplier sent; it is not authority to change a payee or release money. That distinction determines the architecture before any model is chosen.

Stage	What may remain uncertain	What must be established before advancing
Intake	Reading order, document type, ambiguous text	Original retained; each candidate field linked to its location
Validation	Whether an exception has a legitimate business explanation	Currency, line arithmetic, duplicate identifiers and required fields checked
Review	Whether to accept a discrepancy or changed payee	An authorised person sees the original, changes and unresolved checks
Commitment	Whether a remote service has completed an interrupted request	A stable request identifier, permitted transition and recorded external result
Reconciliation	The cause of a mismatch	Local and external records compared; discrepancy handled explicitly

Now introduce a realistic failure: the payment service accepts the instruction, but the network times out before the application receives confirmation. “Try again” is no longer an innocent interface action. The application must resolve the status of the existing instruction or retry under a supported idempotency contract—a rule that makes repeated requests refer to the same intended action. A database audit row alone cannot guarantee that an external payment happened exactly once.

The model has no special authority to settle that uncertainty. It may draft an explanation of the timeout, but neither confident prose nor a recalculated invoice total proves whether money moved. This is why a complete service needs reconciliation and an unresolved state, not merely a successful-path demonstration. Its evidence spans the input, interpretation, approval, attempted action and external outcome.

The example also challenges the claim that deterministic code is automatically safe. Correct arithmetic will faithfully total a fraudulent invoice. Perfect duplicate detection will not authorise a new supplier. A secure permission check can still permit an authorised person to make a poor decision. Each check supports a particular claim; dependable design records where that claim ends and the next responsibility begins.

The construction loop

The practical loop is specification, generation, execution and evidence.

What returns to the next iteration

Rejected proposal

Keep the failed obligation, input and observed result.

Accepted change

Update the durable specification and evidence alongside the implementation.

First, define the outcome and the invariants. What must always hold? Which states are legal? Which actor may perform each transition? What data must exist? What response time and failure behaviour are acceptable?

Second, let AI expand the implementation space. It can propose schemas, services, interfaces, migrations, tests and documentation. This is where probabilistic generation pays for itself: many candidate solutions can be explored quickly.

Third, execute mechanical gates. Compile the code. Validate the schema. Run examples, properties, mutations, accessibility checks, security analysis, performance tests and deployment rehearsals. A gate must be capable of failing for the defect it claims to detect.

Fourth, preserve the evidence. The durable product is not only source code. It includes the specification, decisions, migrations, tests, build records, operational telemetry and handover needed to explain and change the system later.

The loop does not make software infallible. It makes claims bounded and challengeable.

Acceptance should attach to the artefact that was tested. A report saying “all checks passed” becomes ambiguous if the code, dependencies, fixtures or configuration can change before release. Record the relevant revisions and environment, and preserve the output of the actual gate. This does not require archiving every transient log forever. It requires enough provenance to identify the claim, reproduce an investigation and distinguish a tested release from a nearby untested state.

Failure needs an explicit route through the loop. A failed gate can reveal an implementation defect, an invalid test, an inconsistent specification or a newly discovered requirement. Those are different diagnoses with different owners. The worker should not improve its success rate by treating all four as permission to weaken the test. Correct the implementation against the existing obligation, or obtain a reviewed change to the obligation and its evidence. The history should show which happened.

What the evidence does and does not support

Research supports several narrow conclusions: language models can produce fluent falsehoods; preference tuning can create agreement-seeking behaviour; long context can be used unevenly; and machine-generated proposals can be valuable when paired with evaluation. It does not support a universal claim that AI-generated code is unsafe, or that deterministic code is correct by definition.

Two studies, two task boundaries

Copilot study

One bounded programming exercise; faster completion in that setting.

METR study

Experienced developers on familiar repositories; slower with the tested tools.

Productivity evidence is mixed and context-dependent. A [controlled GitHub Copilot task](#) reported faster completion for one bounded programming exercise. A [2025 randomised study by METR](#) found experienced open-source developers were 19 per cent slower with then-current AI tools on familiar repositories, despite expecting a speed-up. Both can be true. They concern different tasks and settings, not a universal comparison of programming with and without AI. Tool value depends on task shape, codebase familiarity, model capability and the cost of verification.

The commercial conclusion is therefore not “AI makes everything ten times faster.” It is that AI moves effort. Less effort may be spent typing implementation; more deliberate effort must be spent defining the target, preparing context and making evidence capable of refusing plausible wrong work.

Productivity comparisons should include the work that moves outside the typing session. Specification preparation, review, rejected patches, integration, incident recovery and operating handover all consume effort. Measuring only time to the first working screen can reward a process that transfers unfinished reasoning to the next person. Conversely, measuring a heavily controlled first deployment without considering later reuse can understate the value of durable gates and domain artefacts.

A sensible local comparison fixes the task boundary and acceptance standard, records the starting familiarity of participants and includes the cost of getting rejected work accepted. Results should be segmented by task type rather than reduced immediately to one average. A model that helps with fixture generation but harms architectural changes is neither universally useful nor useless. The allocation decision is where that evidence becomes economically valuable.

Design consequences

For leaders buying or governing software, five questions expose the boundary:

Separate the product states

Assistant output

An explanation or proposal with its uncertainty retained.

Authoritative output

Validated state, applicable rules and a recorded accountable decision.

- Which answers are generated and which are calculated?
- Can the organisation reproduce a past answer from recorded inputs and a rule version?
- What evidence can reject a release?
- Where is human judgement required, and what information does that person receive?
- Can the organisation change model, supplier or infrastructure without losing its rules and records?

For engineers, the architecture follows. Keep generative components behind explicit interfaces. Treat model output as untrusted input. Validate structure. Put calculations in ordinary functions. Make state machines visible. Preserve provenance. Separate an assistant's explanation from the system's authoritative record.

The objective is not to remove inference. It is to stop inference leaking into places where the product owes an answer.

One implementation consequence is to separate an explanatory view from the authoritative event. A model may produce a clearer explanation of a recorded decision for a particular reader, but that explanation should point to the actual inputs, rule version and result. It must not quietly replace them with newly inferred facts. If the explanation reveals an inconsistency, the product needs a correction or review workflow rather than a more persuasive description of the same record.

Ownership also includes the ability to continue when the model is unavailable. A system of record should not lose access to its rules or historical answers because an assistant endpoint fails. Some generative conveniences may stop; previously committed facts and ordinary deterministic operations should remain available according to the product's service obligation. This makes the boundary tangible to users: the assistant improves how they work with the system, while the organisation's recorded commitments remain its own.

Conclusion

AI is a proposal engine of unusual breadth. Software remains the machinery by which an organisation commits to a repeatable act.

The strongest systems combine the two. They use inference to discover, draft and build. They use explicit specification to define the obligation. They use deterministic computation to produce consequential answers. They use mechanical gates and recorded evidence to show where the claims hold. They leave judgement with people who can be held responsible for it.

Inference makes options. Evidence closes the loop.

References

- Vaswani et al., [Attention Is All You Need](#), 2017.
- Lin, Hilton and Evans, [TruthfulQA](#), 2021.
- Sharma et al., [Towards Understanding Sycophancy in Language Models](#), 2023.
- Liu et al., [Lost in the Middle](#), 2023.
- Anthropic, [Discovering language-model behaviours with model-written evaluations](#), 2022.
- UK Government, [Data and AI Ethics Framework](#), current guidance.
- Peng et al., [The Impact of AI on Developer Productivity: Evidence from GitHub Copilot](#), 2023.
- METR, [Measuring the impact of early-2025 AI on experienced open-source developer productivity](#), 2025.