Vincent Delaroche
Founder of CAST

Executive Summary

This paper examines the economics of using AI to transform complex, enterprise-grade software, and the accuracy and structural integrity of the resulting software.

When AI replaces junior developers on narrow-scoped coding tasks, generates brand-new code, or analyzes small, simple applications, the economics are straightforward. LLMs are cheaper and more productive than human developers in these scenarios. The AI operates within a restricted context, focusing on a bounded portion of code where its probabilistic reasoning works quite well.

But when AI is asked to reverse-engineer very large, custom software systems that function as the "brain of the business" in software-intensive enterprises, the LLMs struggle, and both research and field experience confirm the root cause: AI badly needs architectural context, dependency graphs, transaction and data flows. The need becomes even more pronounced when it comes to transforming a "system of applications", a set of interconnected applications that together support business-critical workflows.

The State of Affairs

The AI industry spends unprecedented sums on raw computing power. Hyper-scalers invested over $370 billion in AI infrastructure in 2025 alone. Global data center electricity consumption is projected to double to 945 TWh by 2030. Why? Because peer-reviewed research from MIT (2025) and others has revealed a striking point: the majority of AI's improvement in recent years has come from throwing more hardware and energy at the problem, not from making algorithms smarter. But more compute doesn't mean more correct. For enterprise software transformation, the gap between 'plausible' and 'right' is where projects fail.

Some will say that LLMs are visibly improving year on year, and providers rightly point to advances in reasoning-focused reinforcement learning, larger-scale pre-training, and new model architectures. These are real. The question is: in what proportion? And the answer supports the thesis of this paper: AI needs smarter, context-aware inputs to deliver on its full potential.

If AI capability is overwhelmingly a function of how much compute you burn, then ungrounded AI — the kind that must discover its environment through step-by-step exploration — is burning the single most expensive resource in the AI economy, while producing accuracy and structural integrity levels too low for enterprise modernization. Ungrounded approaches cost 10× more and deliver results far too unreliable for modernization at scale. Without grounding, LLMs are doing probabilistic archaeology on millions of lines of code, and probabilistic archaeology produces probabilistic results.

The Thesis

The most effective mitigation is not a better prompt or a larger model. It is injecting deterministic intelligence into the equation: real data flows, true execution paths, factual integration points. Information that AI can trust to reason across tens of thousands of interconnected components. Rather than opposing deterministic and probabilistic intelligence, this paper argues that large enterprises should combine the power of AI with the precision and accuracy of deterministic analysis technology, for accuracy and structural integrity that enterprise modernization requires, at a fraction of the cost.

I. The Physics of AI Progress. Compute is (almost) everything

1. The Conventional Wisdom

The popular narrative of AI progress goes something like this: researchers at top labs invent clever new algorithms, architectures, and training techniques. These breakthroughs make models smarter. Computing hardware helps, but the real magic is in the software.

This narrative is, at best, incomplete. A growing body of rigorous, peer-reviewed research demonstrates that the physical infrastructure of AI, GPUs, data centers, and the electricity to power them, has been the dominant driver of capability gains, significantly more than improvements in algorithms or model architectures.

2. The MIT FutureTech Evidence

Two papers from MIT FutureTech provide the most detailed empirical accounting of this dynamic: *"What Drives Progress in AI? Trends in Compute" (January 2025),* and what appears to be a follow up "*On the Origin of Algorithmic Progress in AI" (January 2026).*

This analysis, authored by Slattery, Roded, Del Sozzo, and Lyu at MIT's Computer Science and Artificial Intelligence Laboratory, establishes the foundational case. Drawing on the work of Thompson et al. (2022) and Ho et al. (2024), the paper finds that compute scaling has contributed roughly twice as much as algorithmic progress to effective AI performance gains (Ho et al., 2024). The computing power used to train frontier AI models has increased by roughly 4–5× per year, far outpacing Moore's Law.
Industry actors dominate frontier model development precisely because only they can afford the compute required — the cost of training a single frontier model now runs into the tens or hundreds of millions of dollars. Finally, the relationship between compute and performance follows smooth, predictable scaling laws: more compute reliably produces better models, across domains.

The paper concludes: "*While algorithmic improvements play a vital role, the majority of performance gains have come from increased computational power.*"

This follow-up paper (January 2026) goes further. Rather than estimating the relative contribution of hardware vs. algorithms, it asks a deeper question: Are algorithmic efficiency gains themselves dependent on computing scale? The answer is yes. Through surgical removal of innovations related to algorithms and scaling experiments (testing algorithms across different compute budgets), the authors find that prior estimates credited algorithms with a 22,000× improvement in training efficiency from 2012 to 2023. The new paper can only account for “less than 100×” from specific algorithmic innovations when tested experimentally. The gap is explained by “scale-dependence”: two key innovations (the transition from LSTMs to Transformers, and the shift to Chinchilla-optimal data-parameter balancing) deliver increasing efficiency returns as compute grows. At small compute scales, these innovations provide modest gains. At frontier compute scales, they appear transformative.

The headline finding: “as much as 91% of measured efficiency gains depend on historically exponential compute scaling.” If compute budgets had remained flat, algorithmic innovations alone would have yielded less than 10% of the improvements observed at today's frontier. The paper also introduces a critical concept: “reference-frame dependence”. A sequence of progressively larger (but otherwise identical) transformer models can appear to show continual algorithmic progress relative to an LSTM baseline, even though nothing about the training procedure has changed. The "progress" is entirely an artefact of moving along a scaling curve where the gap to the reference grows with compute. This creates what the authors describe as "the appearance of algorithmic progress without new innovations."

Conclusion: “The algorithms aren't smarter. The hardware makes them look smarter”

It seems severe, but the MIT findings are not isolated. They are consistent with a wide body of research:

- Richard Sutton's "The Bitter Lesson" (2019). The influential AI researcher argued from 70 years of AI history that "general methods that leverage computation are ultimately the most effective, and by a large margin." Methods leveraging human domain knowledge consistently lose to methods that scale with compute.

- OpenAI's "AI and Compute" analysis documented that compute used to train frontier models increased by roughly 10× per year, and that "within many current domains, more compute seems to lead predictably to better performance."

- Harvard Kempner Institute, "The Power of Scale in Machine Learning" (2025) showed that GPT-3 was 100× larger than GPT-2 but based on the same architecture. The capability gains came mostly from scale, not algorithmic big novelty.

- Toby Ord, "Evidence that Recent AI Gains are Mostly from Inference-Scaling" (2025): Even for cutting-edge reasoning models, 82% of performance gains on MATH benchmarks came from inference-scaling (spending more compute at test time), not from algorithmic breakthroughs.

- Epoch AI, "Can AI Scaling Continue Through 2030?" (2024): Projects GPU production expanding 30–100% per year through 2030, identifying GPU cluster growth as the main driver of continued AI progress.

3. The Capital Markets & Electricity Providers Agree

The market itself reveals where value is being created. The single biggest corporate winner of the AI era is not an algorithm company — it is NVIDIA. Hyper-scalers on their side tell the same story. Amazon, Microsoft, Google, and Meta collectively spent over $200 billion on AI infrastructure capex in 2024, with projections exceeding $330 billion in 2025.

On the energy side, the International Energy Agency projects global data center electricity consumption will double to ~945 TWh by 2030; about twice the total energy consumed in France in 2024. The U.S. Department of Energy reports data center electricity climbed from 58 TWh (2014) to 176 TWh (2023), projected to 325–580 TWh by 2028.

II. The Economics of AI Grounding. The agent tax vs the grounding discount

The cost of ungrounded AI manifests in two dimensions. The first is economic: wasted tokens, redundant exploration, runaway compute. The second, seen in Part III, is more consequential: degraded accuracy and compromised structural integrity in the software that results.

1. The Exploration Tax

When an AI agent operates on a large, complex, multi-technology codebase without deterministic architectural context, it must discover the system structure through exploration, assumptions, question-asking, and trial-and-error. Every exploratory prompt, every context reloads, every hallucination-driven false starts, every iteration cycle consumes tokens. AI agents face an exponential discovery problem, which can be described simplistically as follows:

a) AI examines a component and identifies unknown dependencies.
b) AI investigates the most promising dependencies, discovering additional unknowns per dependency.
c) The pattern repeats, creating a discovery tree that can reach hundreds of nodes.

What should be a focused task becomes a "mapping expedition" consuming potentially millions of tokens, with accuracy degrading along the way due to the probabilistic nature of each loop, essentially building probability on probability on probability. It is an accuracy problem. Each probabilistic inference becomes the foundation for the next. By the time the agent has chained together dozens of inferred dependencies, the cumulative confidence is vanishingly low. The token waste and the accuracy loss share the same root cause: the agent is guessing, and each guess compounds the last.

2. Quantifying the Waste

The few empirical studies available — primarily from SWE-bench — measure agents on well-structured open-source Python repositories of 10K–100K lines. Enterprise modernization targets are fundamentally different: larger (often 1M–10M+ LOC), multilingual (Java, COBOL, C, frameworks, SQL, PL/SQL), poorly documented, with decades of accumulated technical debt, misleading comments, circular dependencies, and naming conventions that defeat keyword-based exploration.

These studies focus on token consumption. But the economics extend beyond token cost: the real expense is the failures and rework caused by insufficient accuracy and integrity. Without grounding, each probabilistic inference compounds the next, and hallucination-driven rework accumulates across iterations. The most important waste is not the tokens themselves, but a level of accuracy so low that AI-only modernization projects are increasingly seen as failures.

To make this paper difficult to challenge, the estimates presented below are deliberately conservative. They extrapolate from measured data on simpler codebases and are intended to establish an order of magnitude. Readers are invited to consider that real-world enterprise conditions are likely to push actual consumption above these estimates, not below them.

2.1 Reverse-Engineering and Refactoring

2.1.1 First Layer: Reverse-Engineering

This layer covers the analytical work of understanding each module's architecture, dependencies, data flows, and business logic. It is performed once per module, typically ~ 50 modules for a JavaScript/Java/SQL mid-size enterprise application made of 500K LOC.

Phases	What the Agent Does	Without Grounding (# Tokens)	Without Grounding (Derivation)	Without Grounding (Accuracy)	With Grounding (# Tokens & Accuracy)
1. Initial Discovery (see (1))	Grep, cat/head on entry points, configs, build files to identify modules and layers	300,000	Agent reads ~5–10% of codebase in first pass. 5–10% of 6M = 300K–600K raw code tokens.	Module boundaries assumed. Misidentified modules cascade through all later phases	Tokens: N/A (see (2)) Accuracy: Deterministic analysis
2. Dependency & Import Mapping	Traces import chains, reads pom.xml / build.gradle, follows class hierarchies, framework configs (Spring, EJB)	250,000	Hundreds of files. Agent must read build configs + follow chains. Estimated 4–8% of codebase re-read with cross-referencing across files.	Inferred from keyword/pattern matching : Misses framework-wired dependencies	Tokens: N/A Accuracy: Deterministic analysis
3. Data Flow & Schema Analysis	Reads SQL DDL, DML, entity mappings, JDBC/JPA calls, data transformation logic	100,000	SQL is token-dense. A 500K LOC app may have ~ 50K lines of SQL code if read fully. Agent reads ~20% across multiple passes.	Entity relationships guessed from code patterns: orphaned joins and stored procedure side-effects go undetected	Tokens: N/A Accuracy: Deterministic analysis
4. Business Logic & Transaction Tracing	Follows call chains through controllers → services → repositories. Traces transaction boundaries and business rules.	500,000	Heaviest phase. Agent must trace end-to-end flows across many files, re-reading as it discovers cross-cutting concerns. SWE-bench agents average 51 rounds for 10k line of Python; most rounds are read-heavy exploration.	Inferred, not verified: End-to-end flows are stitched from fragments.	Tokens: ~ 30,000 (to read outputs - call graph, paths, etc) Tokens: N/A Accuracy: Deterministic analysis
5. False Starts & Dead Ends	Exploring deprecated code, wrong branches, misidentified patterns, hallucinated assumptions that require backtracking	400,000	Some runs consume 10× more tokens than others on the same task. Complex agents consume 5–20× more tokens than simple chains due to loops/retries. Conservatively: 10–20% of total effort is wasted exploration.	Hallucinated assumptions propagate downstream: Agent may build entire refactoring plans on incorrect structural premises	Tokens: N/A Accuracy: Deterministic analysis

6. Context Reloading (Accumulated)	Each round re-sends full conversation history as input tokens. Over 40–80+ rounds, prior context compounds.	750,000	If average context per round grows to ~100K tokens by session, and agent runs 50–80 rounds, cumulative re-sent input. This is the "silent killer"	Each reload risks "context drift" : Agent subtly reinterprets earlier findings, introducing silent inconsistencies across rounds	Tokens: N/A Accuracy: Deterministic analysis
7. Human Clarification Loops	Developer explains what agent couldn't discover; agent re-processes affected code regions with new understanding	250,000	Each human correction triggers partial re-exploration. Estimated 5 major clarifications per task, each causing agent to re-read 50K–100K tokens of code with new context.	Agent lacks the architectural scope to propagate a fix to all affected areas, leaving partial inaccuracies	Tokens: ~ 50,000 (Mostly eliminated Assuming 50K tokens for edge cases)
8. Validation & Cross-Referencing	Agent re-reads code to verify its own findings; checks consistency of its architectural model	300,000	Agent has no persistent memory across rounds— must re-read to confirm earlier conclusions. Estimated 2–3 validation passes over key modules (~5% of codebase each).	Agent validates against its own inferred model, not ground truth: Circular verification catches obvious errors but misses structural ones	Tokens: ~ 50,000 (Drastically reduced. Assuming 50K tokens for edge cases)
Tokens used *% reduction*		2.850.000	← up to 5x variance.		150,000 *94.7%*
Accuracy				30% - see (3)	85%+

(1) Note on discovery phase: The general rule of thumb across sources says that 1 token ≈ 4 characters for English text, but code is denser, closer to 3–4 characters per token depending on language verbosity. Static token count of the full 500K LoC codebase would be in the 6 million tokens (read operations alone). The agent doesn't read the whole thing in the discovery phase, as the context window (200K effective for 1M nominal) won't hold it. Hence the agent reads selectively about 5–10% of the codebase, 250K–500K tokens of raw code, plus prompts, history, etc. 500K tokens for "rough" discovery seems to be the right ballpark, 300K to be conservative. To be noted, as the agent can never "see" more than a small chunk of the application at any given moment, it works through a keyhole, round after round, re-reading fragments and trying to stitch together a mental model it can't actually hold.

(2) With grounding, the agent receives the component map, knows exactly which files to touch, understands dependencies upfront. Similar to the SWE-Pruner finding of 23–54% reduction, but grounding goes further than pruning: it eliminates exploration entirely for the architectural context portion (pre-computed actual architecture, dependency graphs, call flows, end to end transactions from user entries to data access) . A conservative estimate should be in the 100K–150K tokens range, per refactoring task,

(3) Accuracy loss compounds across phases. A misidentified module boundary in Phase 1 leads to incorrect dependency mapping in Phase 2, which produces wrong data flow analysis in Phase 3, which corrupts the business logic tracing in Phase 4. By the time the agent reaches validation, it is checking its work against its own flawed model — not against reality.

2.1.2 Second Layer: Refactoring

A complete refactoring is a set of tasks, with a "task" being a single Jira ticket or pull request , one specific change/feature/fix (e.g., "extract payment validation logic into separate service", "refactor error handling in checkout flow", etc.

These tasks fall into several categories:

- Structural refactoring. Extracting monolithic classes into services, breaking circular dependencies, separating concerns (e.g., business logic mixed into controllers or DAO layers), extracting shared utilities, reorganizing package structures. On a 500K LOC app, you might have hundreds of these.

- Data layer modernization. Migrating to modern persistence framework, restructuring entity relationships, normalizing denormalized schemas, extracting repository patterns, replacing stored procedures with application logic (or vice versa), migrating from one database to another.

- API and integration refactoring. Implementing proper API mechanism, standardizing error handling across endpoints, replacing synchronous calls with event-driven patterns, etc.

- Framework and infrastructure upgrades. Migrating from older versions, replacing deprecated libraries, updating authentication/authorization patterns, containerization prep, replacing custom logging/config with standard frameworks.

- Cross-cutting concerns — standardizing exception handling, adding proper observability, security hardening, replacing hardcoded configurations with externalized config, introducing proper dependency injection where it's missing, etc.

- A rule of thumb in enterprise modernization is roughly 1 task per 500 lines of code, knowing that some areas need heavier refactoring, others are just touch-ups. That gives 1,000 tasks for 500K LOC, which aligns with large GSIs typical scoping/pricing for modernization programs.

The 1,000 tasks estimate is at the low-end of the range, especially considering that each of these is a discrete, reviewable pull request.

2.2 Estimating the Average Number of Tokens per Task

Claude Code documentation mentions that a developer consumes roughly 300K–600K tokens per basic (coding tasks on familiar, small-sized codebases) task. SWE-bench baseline on its side shows ~1M token per task for small (10k LoC), well-structured Python repos. For enterprise Java/SQL refactoring tasks, the agent still needs to re-discover context each time, deal with false starts, context reloading, etc. A realistic minimum per-task refactoring estimate should be in the 1.5M–2M tokens range. This per-task figure used in this model is also deliberately conservative. The SWE-bench baseline of ~1M tokens per task was measured on what are arguably the easiest conditions an AI agent could encounter. Even under these favorable conditions, the agent consumed 693K tokens in read operations alone (911K tokens total over about 51 interaction rounds, Sonnet 4.5) just to try to fix a single bug.

The ICLR 2026 finding of up to 10× variance between runs on identical tasks reinforces the idea that a task that averages 1.5M tokens could easily reach 5–10M on a bad run. Broadly speaking, industry reports indicate that agentic workflows have multiplied token consumption per task by 10–100× since late 2023 (Adaline Labs, 2025). Hence the 1.5–2M estimate is one that enterprise practitioners may find optimistic.

3. Translating Tokens to Dollars (and Business Value)

Enterprise-grade modernization warrants frontier models, so using the above estimates and Opus 4.6 pricing ($15/$75 per million input/output tokens), and a blended I/O token cost (80/20 Input/Output), a 1,000-task modernization program on a 500K LOC Java/SQL application would cost ~$43,000 in token consumption alone without architectural grounding. With deterministic grounding, the same program would consume ~125 million tokens at a cost of ~$3,300, for a saving of ~$40,000 and 90+% tokens.

Activity	Without Grounding (# Tokens)	Without Grounding (Cost @ $27/M*)	With Grounding (# Tokens)	With Grounding (Cost @ $27/M*)	Savings
Reverse-engineering (30 modules × per-module estimate)	85.5M	$2,309	4.5M	$122	$2,187
Refactoring (1,000 tasks × per-task estimate)	1.5B	$40,500	120M	$3,240	$37,260
Total program *% token reduction*	1.59B	$42,809	124.5M	$3,362	$39,447 *92.1%*
Accuracy	30%		85%+		

(*) Blended (80/20 Input/Output) cost, in line with SWE-bench findings regarding the Input/output balance.

Worth noticing, without grounding, Reverse-engineering and refactoring are not cleanly separated in practice. What happens is the agent receives a task (e.g., "extract payment validation into a separate service"), and starts exploring the codebase to understand the relevant area, then tries to make the code changes, then explores more to validate dependencies, etc. Economically, this means that for every dollar spent on actual code changes, three dollars are spent on rediscovering architectural context. With grounding, reverse-engineering is done once, outside the agent, by CAST Imaging or other deterministic analysis engine. The pre-computed architecture becomes the input to every refactoring task.

4. Runaway Agent Problem. And It Gets Worse

The scenarios above assume human supervision. When organizations deploy autonomous AI agents without deterministic grounding, economics may become dangerous:

- Infinite exploration loops: Agent discovers a dependency, explores it, finds more dependencies, continues indefinitely. Can consume 1+M tokens before human intervention.

- Hallucination spirals: Agent makes an incorrect assumption, builds on it, receives contradictory feedback, tries to reconcile, makes a new wrong assumption. No convergence means unbounded token consumption.

- Test-fail loops: Agent generates code, tests fail, agent assumes misunderstanding, re-explores entire codebase, tries again, fails again. It can burn millions of tokens before termination.

Each of these failure modes doesn't just burn tokens. It produces code built on compounding errors, amplifying the accuracy crisis described in part III, below.

Modern agentic frameworks now include guardrails — token budgets, iteration caps, loop detection, and cost circuit breakers — that prevent the most extreme runaway scenarios. Guardrails and cost-control mechanisms have emerged since 2025 to prevent runaway costs. Agentic frameworks (LangChain/LangGraph, CrewAI, etc.) now let you set a maximum number of "thought steps" per agent run… and Anthropic or OpenAI enforce requests-per-minute and daily/weekly quotas, specifically because heavy users of Claude Code were burning tokens at unsustainable rates. These mechanisms are necessary and welcome. But they are damage-limiters, not solutions. Guardrails cap the ceiling on waste; grounding eliminates the cause of waste.

III. The Real Risk.
Why accuracy is the decisive factor, not cost

Cost savings matter — but they are not why modernization projects fail. The real case for deterministic grounding is accuracy and structural integrity.

Without architectural grounding, AI agents operating on enterprise-grade applications are, by design, “guessing”. They infer dependencies rather than knowing them. They assume standard patterns where legacy bypasses exist. They hallucinate service boundaries that don't match reality. The LLMs themselves, when prompted, will tell you this directly: they need deterministic architectural intelligence (real component maps, verified dependency graphs, factual execution paths, accurate data access…) to reason reliably about complex systems.

Google's Gemini, for instance, refers to the need for an "Architectural Cheat Sheet." In short, they ask to be grounded, or to be context aware. Once that context is provided, the difference in accuracy and integrity is measurable. Without grounding, AI accuracy on complex enterprise applications hovers around 30% according to several real-life projects. With deterministic architectural context injected into the process, that figure rises above 85% according to a leading global systems integrator who has tested this across multiple enterprise-grade applications.

This matters because in enterprise software modernization, accuracy is not a spectrum. It is a threshold: a refactored module that is 85% correct is not 85% useful. If the remaining 15% includes missed dependencies, broken transaction boundaries, or severed data flows, the module may be worse than what it replaced.

An AI-only approach may also introduce vulnerabilities, break business logic, or silently corrupt data. In regulated industries - banking, insurance, healthcare, aerospace, etc - a single undetected structural defect can trigger audit failures, compliance violations, or operational incidents with costs that dwarf any token savings. Structural integrity compounds this concern. Even when individual code changes appear correct in isolation, the cumulative effect of hundreds of AI-generated modifications on a complex system can degrade system-level qualities — resilience, security, performance — in ways that only become visible under load or in production. Deterministic analysis running against established standards (CWE for security-resiliency-efficiency vulnerabilities, ISO 5055 for the overall system-level structural quality) provides the verification layer that probabilistic AI simply cannot offer on its own. Without it, every AI-generated change is an unverified bet on structural soundness, which means the burden of human verification grows exponentially.

The MIT papers help reframe the hallucination problem. LLMs are probabilistic systems. When asked to deduce architectural relationships from code, they infer rather than analyze. The 2026 MIT paper shows that algorithmic improvements haven't fundamentally changed this. What has improved is that more compute makes probabilistic outputs look more plausible. But while "more plausible" or “very plausible” is fine in numerous areas, in software engineering, “more plausible” is not "correct."

So while a ~$40,000 saving on a modernization program is welcome, the more fundamental question is: what is the cost of modernizing a mission-critical application incorrectly? The answer, for most enterprises, is orders of magnitude higher than the token budget. Deterministic grounding doesn't just make AI cheaper: it makes AI trustworthy enough to use on the systems that matter most.

Conclusion

At 30% accuracy, a 1,000-task modernization doesn't produce a modernized application — it produces a liability. Every undetected broken dependency, every severed data flow, every silently corrupted transaction is a defect that will surface in an audit, in production, or in a security breach.

This is not a cost problem. It is an integrity crisis.

Economics compound the case. Without grounding, a 1,000-task modernization program burns approximately $43,000 in tokens — 92% of which is spent on the agent trying to understand what it's looking at, not on changing code. With deterministic grounding, the same program costs ~$3,300.

Deterministic grounding changes the equation entirely. Not by making the AI smarter, but by giving it something no amount of compute can generate on its own: the truth about how a system actually works. Real dependency graphs. Verified execution paths. Factual data flows. When the AI knows instead of guesses, accuracy rises above 85%, token consumption drops by an order of magnitude, and modernization becomes a controlled engineering process rather than an expensive probabilistic experiment.

The choice for enterprises is not whether to use AI — that question is settled. The choice is whether to let AI explore blindly, burning the scarcest resource in the technology economy on hallucination and rework, or to ground it with deterministic intelligence and direct every token toward productive work.

One path leads to the 40% of agentic AI projects that Gartner predicts will be abandoned by 2027. The other leads to AI-driven modernization that actually delivers — on budget, on time, and with the structural integrity that mission-critical systems demand.

About The Author

Vincent Delaroche
Founder of CAST

A passionate entrepreneur and industry thought leader, Vincent has grown CAST from a French-basement startup to a global market category leader.

Vincent is a true believer in "you can't manage what you don't see". He started a movement, which led to creating the Software Intelligence market category, to help application owners make fact-based decisions, control software risks, accelerate modernization, cloud, and the adoption of AI for across the software development lifecycle.

In CAST, he is building a long-term venture, with the mission to empower humans and AI with the intelligence they need to understand, improve, and transform their software.

Vincent trained in Mechanical Engineering and Computer Science. He loves his family, business and supports children education charities in the US and France. He spends his free time sailing the Atlantic.

Sources:

1) "On the Origin of Algorithmic Progress in AI."

2) "What Drives Progress in AI? Trends in Compute." MIT FutureTech.

3) Ho, A. et al. "Algorithmic Progress in Language Models." Epoch AI / MIT FutureTech, arXiv:2403.05812 (2024).

4) Cornell University – "Tokenomics: Quantifying Where Tokens Are Used in Agentic Software Engineering" . 2026

5) "The Importance of (Exponentially More) Computing Power." arXiv:2206.14007 (2022).

6) Sutton, R. "The Bitter Lesson." incompleteideas.net (2019).

7) OpenAI. "AI and Compute." openai.com/index/ai-and-compute (2018, updated).

8) SWE-Pruner: Self-Adaptive Context Pruning for Coding Agents. Jan 2026.

9) "How Do Coding Agents Spend Your Money? Analyzing and Predicting Token Consumptions in Agentic Coding Tasks" ; https://openreview.net/

10) ICLR 2026 OpenHands study

11) Token Cost Trap: Why Your AI Agent's ROI Breaks at Scale (and How to Fix It). Klaus Hofenbitzer. Nov 2025

12) Kempner Institute, Harvard University. "The Power of Scale in Machine Learning." (2025).

13) Stripe Press. "The Scaling Era: An Oral History of AI, 2019–2025." (2025).

14) Ord, T. "Evidence that Recent AI Gains are Mostly from Inference-Scaling." tobyord.com (October 2025).

15) Epoch AI. "Can AI Scaling Continue Through 2030?" epoch.ai (August 2024).

16) International Energy Agency. "Energy and AI" special report. iea.org (2025).

17) U.S. Department of Energy / Lawrence Berkeley National Laboratory. "2024 Report on U.S. Data Center Energy Use." (2024).

18) Goldman Sachs Research. "AI to Drive 165% Increase in Data Center Power Demand by 2030." (February 2025).

19) NVIDIA Corporation. Quarterly and Annual Financial Results, FY2025–FY2026. nvidianews.nvidia.com.

20) Pew Research Center. "What We Know About Energy Use at U.S. Data Centers Amid the AI Boom." (October 2025).

21) Harvard Belfer Center. "AI, Data Centers, and the U.S. Electric Grid: A Watershed Moment." (February 2026).