mAI-platform

01.03.2026

Beyond the Demo

Why AI-Generated Software Fails in Production - and How to Fix It

\Kristof Hackethal, Founder of mAI

"A whisper began to spread within Silicon Valley that generative AI was not actually useful. The products were falling far short of expectations[...]. Was this just another vaporware cycle?"

\SEQUOIA Capital (Huang & Grady, 2023)

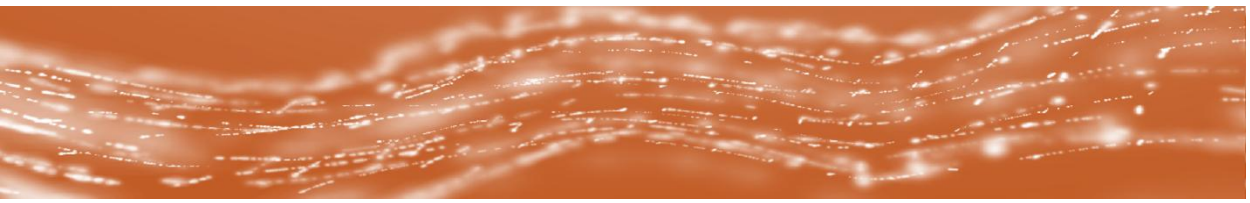
Executive Summary

We are in the transition from AI research to AI implementation. But while the market is flooded with no-code tools – often technologically simple wrappers around proprietary models – these solutions frequently fail in practice at the "last mile," casting significant shadows in the areas of compliance and security. mAI positions itself here with the **APA framework (Application Programming Application)** as a methodical alternative that does not ignore the inherent limitations of generative AI, but compensates for them with structured system architecture.

The core problem with current AI code generators lies in a fundamental contradiction: large language models are probabilistic prediction systems that optimise for statistical plausibility, but software development requires architectural decisions and systematic requirements engineering. Without external orchestration, LLMs develop structural biases and get lost in the complexity of nested dependencies. The economic risk is not primarily in faulty code, but in code that violates security principles or fills conceptual ambiguities with assumptions instead of asking users for clarification.

The APA framework addresses these deficits with a partially deterministic architecture that implements complex requirements in line with proven software engineering principles such as separation of concerns. Research shows that the problems are not technological teething troubles, but systematic consequences of the autoregressive training methodology – they cannot be overcome by scaling, but require structural guardrails at the system level. The APA framework implements precisely this meta-level, transforming AI coding from a stochastic random product into a serious, scalable architecture tool.

Structural Limitations of LLM-based Software Generation



Jensen Huang (CEO, NVIDIA) aptly described how we are transitioning from a decade of AI research to a decade of AI implementation (Huang, 2025). To understand why AI-generated software often fails at the "last mile," it is necessary to consider the fundamental nature of the underlying models.

1.1 Probabilistic Prediction Instead of Deterministic Logic

LLMs maximise the probability of the next token based on previous context (McCoy et al., 2024a). They optimise for *plausibility*, not *correctness* – a distinction that is fundamental to software generation. Once a token is set, it is effectively fixed. Unlike human engineers who can iterate and replan, a stand-alone LLM largely lacks this meta-level (Bubeck et al., 2023). It generates code strictly from left to right with no possibility of revising earlier decisions – a commitment problem.

Newer approaches such as reasoning models partially address this limitation but do not completely overcome it, as they continue to show high sensitivity to statistical probability from training data (McCoy et al., 2024b). Without external quality control, LLM-generated solutions tend to reproduce frequently seen patterns. They are powerful pattern completion engines that lack the scaffolding for principle-guided reasoning (Zhang et al., 2025).

1.2 LLMs as Collective Intelligence – and Its Limits

LLMs can be understood through the analogy of collective intelligence: they aggregate knowledge encoded during training (Petroni et al., 2019). Like collective intelligence, they outperform individuals on many standardised benchmarks (Bubeck et al., 2023; OpenAI et al., 2024), though interpretation remains contested (Martínez, 2025). However, this analogy also reveals structural weaknesses:

\Overrepresentation instead of diversity Just as collective intelligence fails when diversity is lacking, LLMs are subject to biases from overrepresentation in training data (Burton et al., 2024). Architectural decisions reflect statistical dominance of certain patterns rather than best practices.

\Limits of combinatorial creativity Generative AI innovates primarily through novel recombination of existing concepts (Franceschelli & Musolesi, 2025b). Outputs tend towards common patterns rather than transformative deviations (Franceschelli & Musolesi, 2025a).

\Lack of coordination in complex tasks Current research shows that LLMs are fundamentally incapable of autonomous planning and require external orchestration (Kambhampati et al., 2024). Even specialised reasoning models do not saturate established planning benchmarks (Valmeekam et al., 2024).

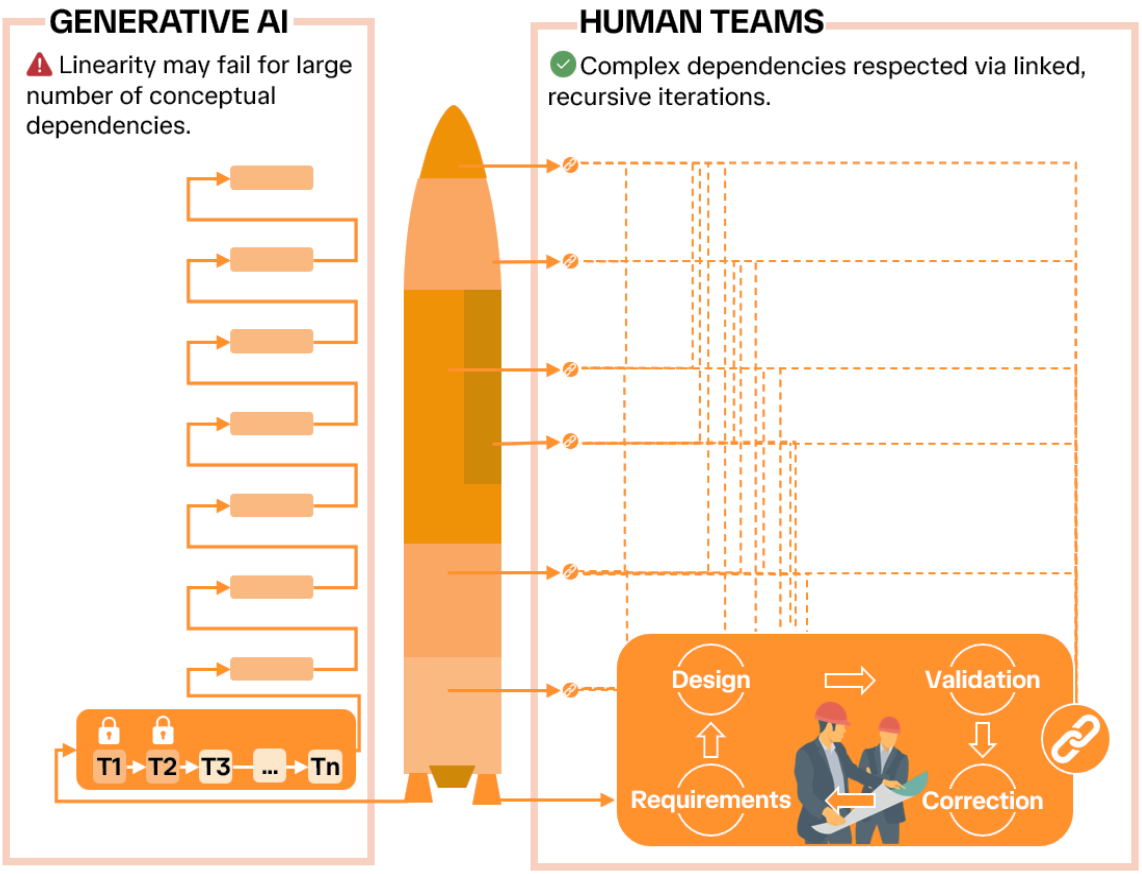


Figure 1: Linearity as a bottleneck

3 Sycophancy and silent assumption formation

The most serious distortion for software generation lies in the overarching optimisation goal: task fulfilment. The alignment process, consisting of instruction tuning and reinforcement learning from human feedback (RLHF), conditions models to be helpful assistants (Ouyang et al., 2022). The downside

is that this completeness pressure leads to assumptions, especially in software generation. For example, if a user requests the calculation of a snowball's flight path, the model will assume air friction and temperature are negligible without the user specifying this.

We refer to this as **silent assumption formation**, where missing specifications are substituted by plausible but unvalidated assumptions. This is related to sycophancy: models adapt responses to the user's presumed opinion even when objectively incorrect (Sharma et al., 2025; Wei et al., 2024). Research shows that model scaling and instruction tuning significantly *increase* sycophancy (Wei et al., 2024). In code generation, this manifests as models treating user queries as complete and correct rather than pointing out gaps. Empirically, code LLMs rarely recognise incomplete or contradictory task descriptions (Larbi et al., 2025), and LLMs process ambiguous requests directly without asking for clarification (Mu et al., 2024).

Consider: "Convert my Excel files to JSON." A human engineer would clarify the structure and desired schema. An LLM fills this gap with assumptions, generating a solution that fails when real data deviates (Chen et al., 2024).

Table 1: Human vs. AI approach to Requirements

Feature	Human Senior Engineer	LLM Code Generator
Handling Ambiguity	Actively asks questions, clarifies constraints	Fills gaps with silent assumptions
Error Culture	Points out risks and knowledge gaps	Confirms the user prompt as complete and correct
Transparency	Makes uncertainties and trade-offs explicit	Generates plausible sounding but unvalidated solutions
Result	Validated requirements	"Illusion of Progress" – hard-to-detect logical errors

4 Further distortions in the generation process

Three additional biases compound the problem:

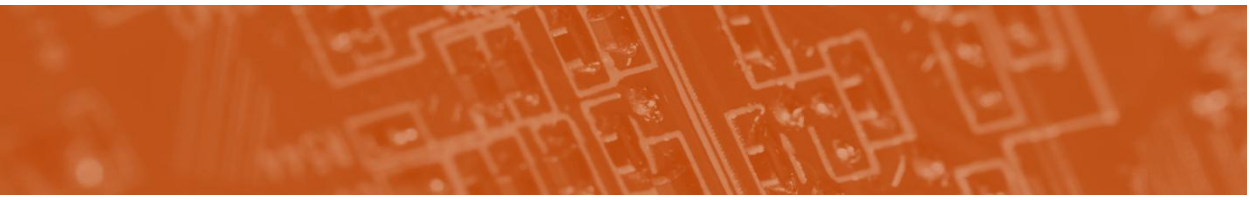
\Position-dependent information weighting LLMs process information at the beginning and end of a context better than the middle – the "lost in the middle" phenomenon (Liu et al., 2024). In code generation, prompt order can influence the resulting architecture.

\Verbosity bias Both human evaluators and LLMs rate longer responses as higher quality (Saito et al., 2023; Zheng et al., 2023). Since this preference enters training via RLHF, models plausibly tend towards more extensive solutions in code generation.

\Lack of self-correction Without external reference points, LLMs cannot reliably detect their own errors, self-correction attempts can even worsen performance (Huang et al., 2024).

These biases act cumulatively: a model that underweights middle requirements, tends towards complex solutions, and cannot recognise its own errors produces compounding architectural shortcomings.

Shortcomings with current AI Builders



1 Illusion of Progress

The biases described in the previous sections are compounded by market dynamics that exacerbate rather than mitigate their consequences. The demand for automated software creation is growing rapidly – in an international survey of 2,000 companies from the EMA, US and ASPAC regions, 81% of respondents considered low-code development strategically relevant to their organisation (KPMG, 2024a). The market is increasingly dominated by solutions advertising promises such as "Apps in 30 seconds" or "from idea to application in minutes." This positioning systematically prioritises speed over validity and creates a dangerous expectation: that the quality of a software product is primarily a function of the speed at which it is generated.

The fundamental problem: user requirements are almost never complete. The CHAOS Report identified incomplete requirements as the most common factor in software project failure (The Standish Group, 1995). Ambiguity in natural language is unavoidable (Berry & Kamsties, 2004), and current LLMs do not ask for clarification, they translate unclear requirements directly into code. Only systematic clarification mechanisms significantly improved quality: GPT-4 achieved 80.80% pass@1 with queries versus 70.96% without (Mu et al., 2024).

This is compounded by **code hallucinations**: missing requirements are filled with plausible but incorrect additions (Ji et al., 2023). Zhang et al. (2025) identified categories including the invention of non-existent functions and

incorrect parameter structures. Liu et al. (2026) documented additional categories including fabricated APIs and data structures.

2 Sequential Decomposition as a Counter-Model

The common approach “from prompt directly to finished code” suffers from an operationalized sycophancy bias: the model assumes that all information is available and correct and generates complete applications on this basis. A solution lies in the sequential decomposition of the generation process, which follows the findings of classic requirements engineering. Following IEEE 29148 (IEEE, 2018), the generation process can be decomposed into four phases:

(1) **Domain Understanding** – analyzing requirements, actors, and core processes

(2) **Clarification** – asking specific questions about ambiguities before code generation

(3) **Data Model Design** – creating a verifiable intermediate representation

(4) **Production-Grade Code Generation** – generating code only after validation of previous phases.

This corresponds to the "slow down to speed up" principle established since Boehm's Spiral Model (Boehm, 1986).

3 The Case for Decoupled Architecture

Most current AI builders rely on monolithic "all-in-one" stacks. While this minimizes initial friction, it reaches systematic limits with complex B2B requirements (Hellerstein et al., 2018). Sustainable automation requires decoupled architecture with strict front-end/back-end separation, for three reasons:

\Security through isolation The US National Institute od Standards and Technology (NIST) states that layered architectures provide hardened layers between exposed web servers and sensitive data (Chandramouli, 2019). In a monolithic stack, these layers collapse.

\Enterprise connectivity B2B software must communicate with legacy systems. Python is the dominant language in data science and ML (GitHub, 2024; Stackoverflow, 2024), and a Python backend enables native library access for ERP middleware.

\AI maintainability Architectural separation improves AI generation quality by reducing the search space. LLMs show significant performance losses with irrelevant context (Shi et al., 2023), degrade with information positioned in longer contexts (Liu et al., 2024), and hallucinate more with complex cross-module dependencies (Zhang et al., 2025). Strict decoupling

lets the generator put on "blinkers" – when changing backend logic, it cannot accidentally alter the frontend layout.

4 The Innovator's Dilemma

If the advantages are clear, why aren't established platforms adapting? Established builders face technological path dependency. Most are built on tightly coupled frameworks that facilitate speed & simplicity but create technical debt (Al Alamin et al., 2021; Alharbi & Alshayeb, 2026). A pivot would be disruptive and cannibalize existing business models.

The argument that larger context windows eliminate the decomposition-need is a technological fallacy. A larger context does not solve attention dilution (Liu et al., 2024). Decomposition is not a limitation due to lack of context capacity, but a conscious decision to maximize precision – giving the model only the absolutely necessary context forces it to focus on the actual.

The principles outlined above, sequential decomposition and architectural decoupling, are established in software engineering but absent from most of today's AI builders. The APA framework operationalize these principles into a concrete, reproducible system.

The APA Framework: A Methodology for Deterministic Software Orchestration

To bridge the gap between stochastic generation and deterministic production readiness, we have developed the **APA** (Application Programming Application)

framework. This approach represents a conscious counterpoint to the prevailing fully dynamic status quo.

The goal is a system whose output remains logically comprehensible and maintainable for both human developers and future AI iterations through deterministic rules, assumes security by default, and achieves flexibility not despite, but because of, a limitation of degrees of freedom. The philosophy of the framework is based on four core principles.

1 First Understand, Then Build

The generation paradigm is reversed: before any code is written, the system must first capture and clarify what needs to be built. The system asks non-technical users the right technical questions without cognitive overload, using dynamic navigation that prioritises and groups questions. Contextual efficiency eliminates redundant questions – if a user specifies "internal tool without customer login," questions about sign-up processes are automatically pruned. Time efficiency is achieved through asynchronous generation: once front-end-relevant questions are complete, partial code generation begins in the background while users continue answering backend-relevant queries, transforming a sequential approach into a quasi-parallel one.

2 Semi-Deterministic Structure

The foundation of APA is the departure from a blank sheet. Critical infrastructure, e.g. password management, encryption, database connectors, is firmly anchored ("hard-coded"). An LLM within APA may choose the wrong colour for a button if not specified, but it *cannot* implement insecure password storage because it does not generate that code, only references it.

APA also orchestrates internal communication logic, specifying communication paths (API calls, serialisation) to manage polyglot architecture complexity. For external system integration, APA accesses an internal database with approximately 56,000 verified API templates via vector search. The HTTP structure is never generated by the LLM. Write operations (POST/PUT/DELETE) undergo a verification loop: APA retrieves the current status of the target resource and presents it to the user for verification before the write transaction is released – preventing "blind flights" at critical points.

3 Semi-Deterministic Alignment

APA resolves the conflict between dynamic user intent and deterministic logic through hierarchical decision trees, based on the FLOW concept (Niu et al., 2025). The process follows a strict sequence of **breadth before depth**, similar to the MECE principle, ensuring structural integrity before working out details.

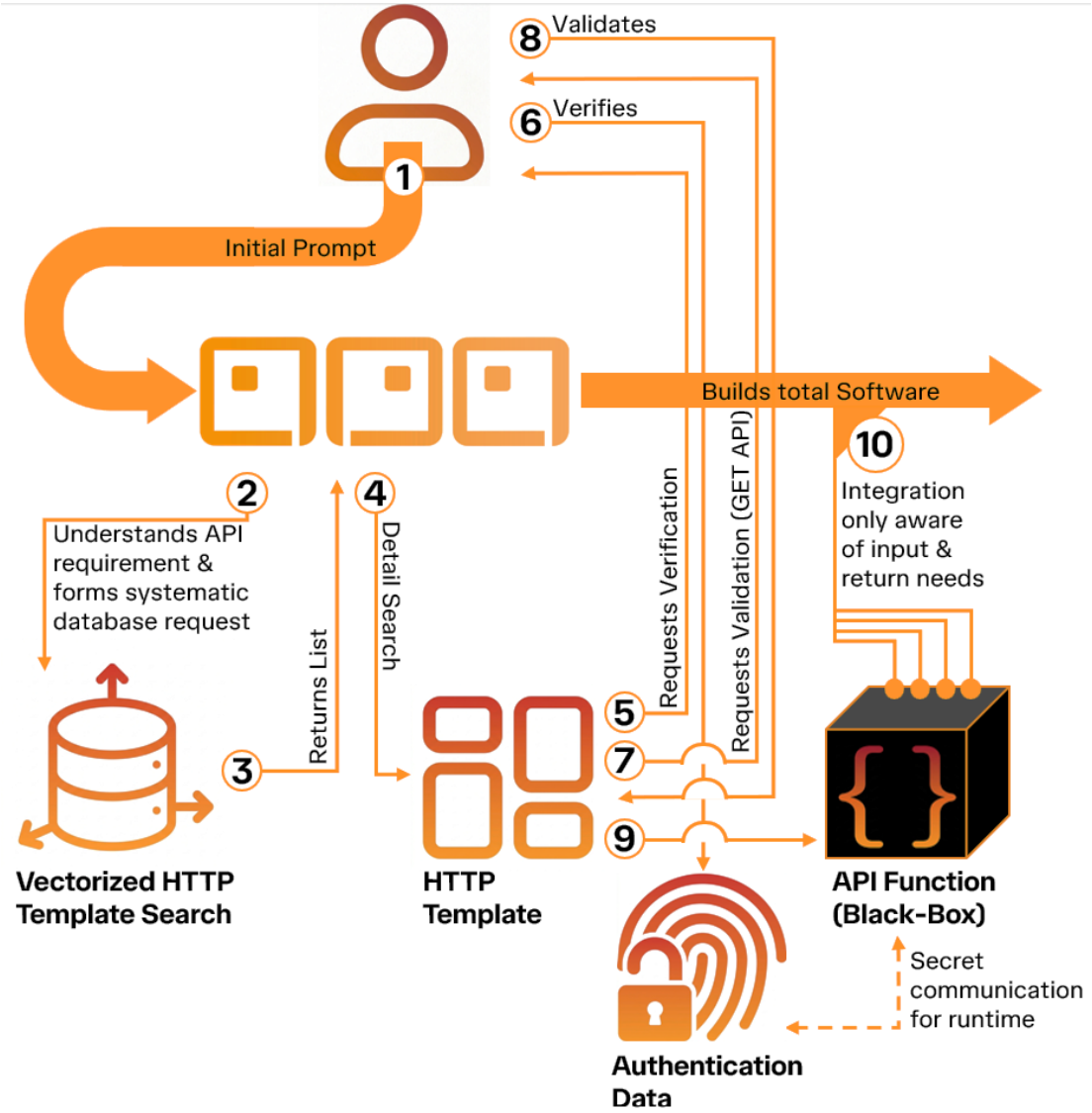


Figure 2: Semi-deterministic API Integration

The decisive advantage lies in **active complexity reduction**: decisions made at one level rigorously prune illogical future paths. If the system selects a "non-interactive" UI element, all branches dealing with onClick logic, backend triggers, or state changes are removed. Associated rulesets are not loaded into subsequent AI agents' context, preventing hallucinations. Pruning acts as a technical filter, the system doesn't rely on AI to "understand" that a text field doesn't need a button click, it removes the possibility of that error entirely. This corresponds with findings that structured tree search methods are necessary for navigating generative workflows efficiently (Zhang et al., 2024).

4 Structural Transparency

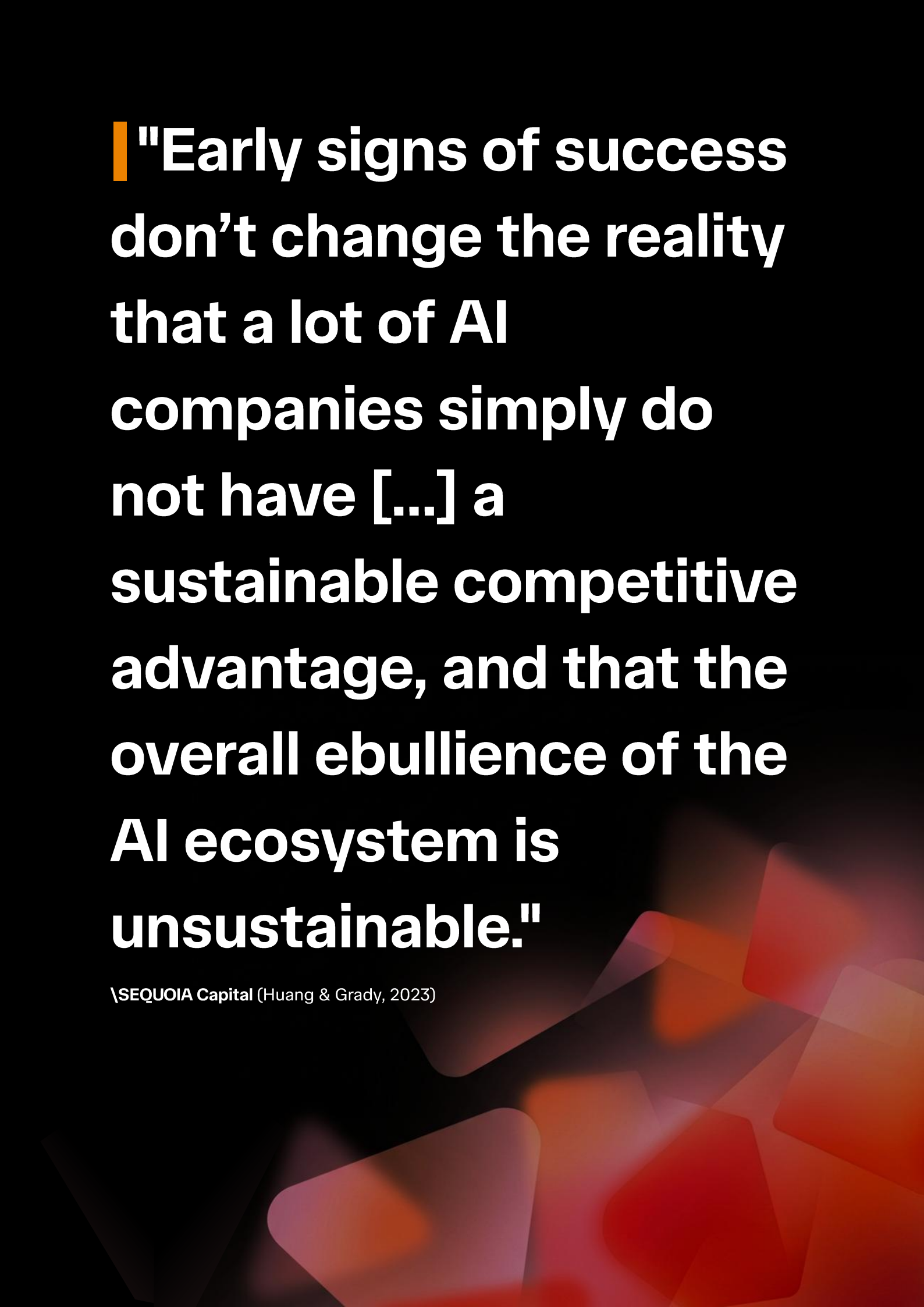
APA implements the principle of self-documenting architecture. Since the architecture follows fixed topological patterns, the system can algorithmically predict where particular logic resides – "searching" is replaced by precise "addressing." File names are functional addresses: e.g.

apa_welcomePage_cmdViewDetails.py encodes the architectural assignment, enabling implicit routing from UI to business logic.

This transparency also reduces cognitive load for human auditors (Fakhoury et al., 2018), ensuring seamless integration into existing business processes by generating structures that are intuitively understandable and standard-compliant.

"Early signs of success don’t change the reality that a lot of AI companies simply do not have [...] a sustainable competitive advantage, and that the overall ebullience of the AI ecosystem is unsustainable."

\SEQUOIA Capital (Huang & Grady, 2023)



Implications for mAI

The technical necessity of deterministic orchestration is not an end in itself, it is the prerequisite for a fundamental shift in software-economic reality. When software generation is no longer stochastic and fragile but structured and maintainable, the strategic position of the companies using it changes.

\The wrapper problem The current market is characterised by thin wrappers over models of a few US providers, creating significant platform risk (Appenzeller & Li, 2025; Chandrasekaran, 2025). Proprietary model providers hold the strategic leverage to drive competitors out of the market by adjusting API costs or expanding functionality (Huang & Grady, 2023). Companies that base their digital value creation entirely on these providers face both economic and technological dependency. The alternative is to treat LLMs as an interchangeable engine within a fixed architectural chassis, reducing dependency to a replaceable component rather than a structural foundation.

mAI therefore clearly positions itself as an infrastructure partner for **autonomous enterprise innovation**.

Strategic Implications for Customers

\The reversal of build-or-buy The drastic reduction in marginal costs in code production inverts traditional economic logic. An average medium-sized company has around 130 SaaS products (mAI Research). Since deterministic frameworks automate the costly "last mile," in-house development becomes more economically rational than generic standard software. Processes previously "too small for IT but too big for Excel" can now be digitised economically, potentially saving thousands of administrative hours per year.

\The scaling gap While 74% of companies report positive outcomes in initial AI pilots, only 31% succeed in scaling these solutions (KPMG, 2024b). With regulatory compliance (38%) and risk management (32%) as the main challenges of generative AI (Deloitte, 2025), and 42% citing security as a core concern for low-code specifically (KPMG, 2024a), tools optimised purely for go-to-market speed risk squandering trust. Closing this gap requires enterprise-ready architecture that understands security and regulatory compliance as an architectural foundation – compliance by design, not compliance as an afterthought.

About mAI

mAI is a prompt-based "Autonomous Enterprise Innovation" platform start-up from Hamburg that enables SMEs and enterprise customers to develop tailor-made business software. mAI delivers production-ready software solutions for companies in minutes that go beyond proof-of-concepts. While the market is dominated by no-code tools that act as simple wrappers around proprietary models, mAI addresses the fundamental challenges of AI-generated software: scalability, maintainability, and enterprise compliance.

To this end, mAI uses its internally developed APA framework (Application Programming Application), which guides users through the interactive recording of user requirements, the integration of a novel product manager functionality to eliminate missing context, and the provision of the final security module for orchestrating and generating secure and maintainable code by the APA architect - all in a guided environment for the best possible user experience. With mAI-APA, companies can develop, test and roll out software in a short period of time. When it comes to deployment, mAI also relies on proven standards: companies do not have to deploy their software in shared spaces, but instead experience a new level of security and trust through individual, company-specific virtual machines (VMs) with servers located in Germany.

At mAI, we know that generative artificial intelligence has limitations that we will not be able to compensate for with more powerful models in the next five years, nor are they ones that we want to unleash unchecked on critical areas of enterprise software. If the security of your processes is important to you, take a second look. Modular, adaptive intelligence (mAI) stands for a solid framework on which you can build and grow.

About the Author

Kristof Hackethal is a technology entrepreneur and AI-focused professional with an industry & academic background in AI strategy, human–AI interaction, and enterprise software architecture. His work bridges academic research and real-world application, informed by research experience at leading institutions such as ETH Zurich.

Legal Disclosure

EXIST Funding

We are proud to be funded by the science-driven EXIST program financed by ...

Contact Us

Reach out to us via team@mai-platform.de

https://mai-platform.com

References

Al Alamin, M. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., & Iqbal, A. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 46–57. https://doi.org/10.1109/MSR52588.2021.00018

Alharbi, M., & Alshayeb, M. (2026). Automatic Code Generation Techniques: A Systematic Literature Review. *Automated Software Engineering*, *33*(1), 4. https://doi.org/10.1007/s10515-025-00551-3

Altrogge, G. (2018). *Netzplantechnik* (3. Aufl. Reprint 2018). Oldenbourg Wissenschaftsverlag. https://doi.org/10.1515/9783486790405

Appenzeller, G., & Li, Y. (2025). *The Trillion Dollar AI Software Development Stack*. Andreessen Horowitz. https://a16z.com/the-trillion-dollar-ai-software-development-stack/

Berry, D. M., & Kamsties, E. (2004). Ambiguity in Requirements Specification. In J. C. S. do Prado Leite & J. H. Doorn (Eds.), *Perspectives on Software Requirements* (pp. 7–44). Springer US. https://doi.org/10.1007/978-1-4615-0465-8_2

Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, *11*(4), 14–24. https://doi.org/10.1145/12944.12948

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4* (arXiv:2303.12712). arXiv. https://doi.org/10.48550/arXiv.2303.12712

Burton, J. W., Lopez-Lopez, E., Hechtlinger, S., Rahwan, Z., Aeschbach, S., Bakker, M. A., Becker, J. A., Berditchevskaia, A., Berger, J., Brinkmann, L., Flek, L., Herzog, S. M., Huang, S., Kapoor, S., Narayanan, A., Nussberger, A.-M., Yasseri, T., Nickl, P., Almaatouq, A., … Hertwig, R. (2024). How large language models can reshape collective intelligence. *Nature Human Behaviour*, *8*(9), 1643–1655. https://doi.org/10.1038/s41562-024-01959-9

Canva. (2024, January). *CIO AI Report Findings*. https://www.canva.com/newsroom/news/cio-ai-report-findings/

Chandramouli, R. (2019). *Security strategies for microservices-based application systems* (NIST SP 800-204; p. NIST SP 800-204). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204

Chandrasekaran, A. (2025). *The 2025 Hype Cycle for GenAI Highlights Critical Innovations*. Gartner Inc. https://www.gartner.com/en/articles/hype-cycle-for-genai

Chen, Q., Yu, J., Li, J., Deng, J., Chen, J. T. J., & Ahmed, I. (2024). *A Deep Dive Into Large Language Model Code Generation Mistakes: What and Why?*

Deloitte. (2025, January). https://www2.deloitte.com/content/dam/Deloitte/us/Documents/consulting/us-state-of-gen-ai-q4.pdf

Fakhoury, S., Ma, Y., Arnaoudova, V., & Adesope, O. (2018). The effect of poor source code lexicon and readability on developers' cognitive load. *Proceedings of the 26th Conference on Program Comprehension*, 286–296. https://doi.org/10.1145/3196321.3196347

Franceschelli, G., & Musolesi, M. (2025a). Creativity and Machine Learning: A Survey. *ACM Computing Surveys*, *56*(11), 1–41. https://doi.org/10.1145/3664595

Franceschelli, G., & Musolesi, M. (2025b). On the Creativity of Large Language Models. *AI & SOCIETY*, *40*(5), 3785–3795. https://doi.org/10.1007/s00146-024-02127-3

GitHub. (2024). Octoverse: AI leads Python to top language as the number of global developers surges. *The GitHub Blog*. https://github.blog/news-insights/octoverse/octoverse-2024/

Glikson, E., & Woolley, A. (2020). Human trust in artificial intelligence: Review of empirical research. Academy of Management Annals (in press). *The Academy of Management Annals*.

Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., & Wu, C. (2018). *Serverless Computing: One Step Forward, Two Steps Back* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.1812.03651

Hong, L., & Page, S. (2004). *Groups of diverse problem solvers can outperform groups of high-ability problem solvers*. https://doi.org/10.1073/pnas.0403723101

Huang, J. (2025, January 27). *NVIDIA CEO Jensen Huang's Vision for the Future (Min 23:08)* [Interview]. https://www.youtube.com/watch?v=7ARBJQn6QkM

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). *Large Language Models Cannot Self-Correct Reasoning Yet* (arXiv:2310.01798). arXiv. https://doi.org/10.48550/arXiv.2310.01798

Huang, S., & Grady, P. (2023). *Generative AI's Act Two* [Industry Report]. Sequoia. https://sequoiacap.com/article/generative-ai-act-two/

IEEE. (2018). ISO/IEC/IEEE International Standard—Systems and software engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, 1–104. https://doi.org/10.1109/IEEESTD.2018.8559686

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, *55*(12), 248:1-248:38. https://doi.org/10.1145/3571730

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., & Murthy, A. (2024). *LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks* (arXiv:2402.01817). arXiv. https://doi.org/10.48550/arXiv.2402.01817

KPMG. (2024a). *Low-code adoption as a driver of digital transformation*. https://assets.kpmg.com/content/dam/kpmgsites/content/dam/kpmgsites/xx/pdf/2024/02/kpmg-low-code-adoption-as-a-driver-of-digital-transformation.pdf.coredownload.inline.pdf

KPMG. (2024b, September). *Global Tech Report*. https://assets.kpmg.com/content/dam/kpmgsites/xx/pdf/2024/09/kpmg-global-tech-report-2024.pdf

Larbi, M., Akli, A., Papadakis, M., Bouyousfi, R., Cordy, M., Sarro, F., & Traon, Y. L. (2025). *When Prompts Go Wrong: Evaluating Code Model Robustness to Ambiguous, Contradictory, and Incomplete Task Descriptions* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2507.20439

Liu, F., Liu, Y., Shi, L., Yang, Z., Zhang, L., Lian, X., Li, Z., & Ma, Y. (2026). *Beyond Functional Correctness: Exploring Hallucinations in LLM-Generated Code* (arXiv:2404.00971). arXiv. https://doi.org/10.48550/arXiv.2404.00971

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, *12*, 157–173. https://doi.org/10.1162/tacl_a_00638

Martínez, E. (2025). Re-evaluating GPT-4's bar exam performance. *Artificial Intelligence and Law*, *33*(3), 581–604. https://doi.org/10.1007/s10506-024-09396-9

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., & Griffiths, T. L. (2024a). Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, *121*(41), e2322420121. https://doi.org/10.1073/pnas.2322420121

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., & Griffiths, T. L. (2024b). *When a language model is optimized for reasoning, does it still show embers of autoregression? An analysis of OpenAI o1* (arXiv:2410.01792; Version 1). arXiv. https://doi.org/10.48550/arXiv.2410.01792

Mu, F., Shi, L., Wang, S., Yu, Z., Zhang, B., Wang, C., Liu, S., & Wang, Q. (2024). ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification. *Proceedings of the ACM on Software Engineering*, *1*, 2332–2354. https://doi.org/10.1145/3660810

Newman, S. (2021). *Building microservices: Designing fine-grained systems.* O'Reiley Media, Inc.

Niu, B., Song, Y., Lian, K., Shen, Y., Yao, Y., Zhang, K., & Liu, T. (2025). *Flow: Modularized Agentic Workflow Automation* (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2501.07834

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2024). *GPT-4 Technical Report* (arXiv:2303.08774). arXiv. https://doi.org/10.48550/arXiv.2303.08774

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). *Training language models to follow instructions with human feedback* (arXiv:2203.02155). arXiv. https://doi.org/10.48550/arXiv.2203.02155

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., & Miller, A. (2019). Language Models as Knowledge Bases? In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 2463–2473). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1250

Reinsel, D., Gantz, J., & Rydning, J. (2018). *The Digitization of the World from Edge to Core*.

Saito, K., Wachi, A., Wataoka, K., & Akimoto, Y. (2023). *Verbosity Bias in Preference Labeling by Large Language Models* (arXiv:2310.10076). arXiv. https://doi.org/10.48550/arXiv.2310.10076

Sharma, M., Tong, M., Korbak, T., Duvenaud, D., Askell, A., Bowman, S. R., Cheng, N., Durmus, E., Hatfield-Dodds, Z., Johnston, S. R., Kravec, S., Maxwell, T., McCandlish, S., Ndousse, K., Rausch, O., Schiefer, N., Yan, D., Zhang, M., & Perez, E. (2025). *Towards Understanding Sycophancy in Language Models* (arXiv:2310.13548). arXiv. https://doi.org/10.48550/arXiv.2310.13548

Shi, F., Chen, X., & Misra, K. (2023). *Large Language Models Can Be Easily Distracted by Irrelevant Context*. ResearchGate. https://www.researchgate.net/publication/367961918_Large_Language_Models_Can_Be_Easily_Distracted_by_Irrelevant_Context

Stackoverflow. (2024). *Technology | 2024 Stack Overflow Developer Survey*. https://survey.stackoverflow.co/2024/technology

The Standish Group. (1995). *The Chaos Report*. ResearchGate. https://www.researchgate.net/publication/263849222_The_Chaos_Report

Turpin, M., Michael, J., Perez, E., & Bowman, S. R. (2023). *Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting* (arXiv:2305.04388). arXiv. https://doi.org/10.48550/arXiv.2305.04388

Valmeekam, K., Stechly, K., & Kambhampati, S. (2024). *LLMs Still Can't Plan; Can LRMs? A Preliminary Evaluation of OpenAI's o1 on PlanBench* (arXiv:2409.13373). arXiv. https://doi.org/10.48550/arXiv.2409.13373

Wang, P., Li, L., Chen, L., Cai, Z., Zhu, D., Lin, B., Cao, Y., Liu, Q., Liu, T., & Sui, Z. (2023). *Large Language Models are not Fair Evaluators* (arXiv:2305.17926). arXiv. https://doi.org/10.48550/arXiv.2305.17926

Wei, J., Huang, D., Lu, Y., Zhou, D., & Le, Q. V. (2024). *Simple synthetic data reduces sycophancy in large language models* (arXiv:2308.03958). arXiv. https://doi.org/10.48550/arXiv.2308.03958

Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., Zheng, B., Liu, B., Luo, Y., & Wu, C. (2024). *AFlow: Automating Agentic Workflow Generation* (Version 4). arXiv. https://doi.org/10.48550/ARXIV.2410.10762

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Yu, Chen, Y., Wang, L., Luu, A. T., Bi, W., Shi, F., & Shi, S. (2025). Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *Computational Linguistics*, *51*(4), 1373–1418. https://doi.org/10.1162/COLI.a.16

Zhang, Z., Wang, C., Wang, Y., Shi, E., Ma, Y., Zhong, W., Chen, J., Mao, M., & Zheng, Z. (2025). LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation. *Proc. ACM Softw. Eng.*, *2*(ISSTA), ISSTA022:481-ISSTA022:503. https://doi.org/10.1145/3728894

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena* (arXiv:2306.05685). arXiv. https://doi.org/10.48550/arXiv.2306.05685