Selling Shovels in the LLM Gold Rush: Why Software Engineering Research Risks Missing the Real Transformation

Önder Babur*, Sébastien Mosser†, and Alfonso Pierantonio‡

*Wageningen University & Research, The Netherlands

†McMaster University, Canada

‡Università degli Studi dell'Aquila, Italy

ABSTRACT The rapid adoption of Large Language Models (LLMs) is reshaping software engineering practice at an unprecedented pace. While research on LLM-based software engineering is flourishing, much of it emphasizes frameworks, roadmaps, and conceptual models that are easy to articulate and evaluate, yet increasingly detached from the realities of sustained, large-scale use. This editorial argues that a growing gap is emerging between how LLMs are studied in software engineering research and how they are actually engineered and operated in industrial settings. Key distinctions related to scale, cost, duration of use, and organizational impact are often left implicit, leading to a focus on tactical performance improvements while overlooking the strategic transformation of software production. We discuss the risks of accompanying and proxy research, examine their implications for research credibility, industrial relevance, and education, and propose concrete steps toward greater proximity between research discourse and engineering reality. Without such proximity, software engineering research risks talking about the transformation rather than helping to shape it.

KEYWORDS Large Language Models, Software Engineering, Research Practice, Engineering Reality

1. Introduction

Research on Large Language Models (LLMs) and Generative AI in Software Engineering is expanding at an extraordinary pace (Hou et al. 2024; Zhang et al. 2025). In a relatively short time, LLMs have moved from being perceived as experimental tools for code generation to becoming pervasive components across the software development lifecycle. From initial surprise, often accompanied by controversy, the community has rapidly transitioned to a form of disbelief at the breadth and sophistication of what these systems are now able to achieve.

LLMs are currently used to support activities such as code completion, test generation, documentation, refactoring, bug fixing, requirements elicitation, architectural exploration, and developer assistance within integrated development environments. In industrial settings, they are increasingly embedded into development workflows, continuous integration pipelines, internal tooling, and organizational knowledge bases (Hou et al. 2024; Zhang et al. 2025).

Yet, this apparent ubiquity conceals an important asymmetry. Not all interactions with LLMs are equivalent. The depth, stability, and significance of observed results depend strongly on the scale, continuity, and cost of their usage. Sporadic, free-tier interactions expose only a narrow and often distorted slice of LLMs' behaviour. Sustained engagement with high-capacity models, long contexts, and large volumes of interaction reveals qualitatively different phenomena, including emergent capabilities, failure modes, and forms of cognitive and technical debt that remain invisible at small scale.

Consider a tangible illustration. A researcher using `GPT-4o` through a free tier encounters rate limits after a handful of queries, works with truncated context windows, and observes

JOT reference format:
Önder Babur, Sébastien Mosser, and Alfonso Pierantonio. *Selling Shovels in the LLM Gold Rush: Why Software Engineering Research Risks Missing the Real Transformation*. Journal of Object Technology. Vol. 25, No. 02, 2026.
http://dx.doi.org/10.5381/jot.2026.25.02.e1

latencies that vary unpredictably. An engineering team paying $20,000–$50,000 per month for API access to the same model family operates in a different regime entirely (Pan et al. 2025): they encounter billing anomalies when prompt lengths drift upward, discover that model updates silently alter output distributions (Chen et al. 2024), and must design retry logic for a service whose failure modes are both frequent and non-deterministic (Chu et al. 2025). These are not incremental differences. They are qualitatively distinct experiences of the same technology, and they produce fundamentally different forms of knowledge.

This asymmetry is rarely acknowledged in research practice. Much academic exploration relies on limited, low-cost access to LLMs, while many of the most consequential transformations are unfolding through engineering-scale and industrial-scale deployments. As a result, research risks accompanying the technology from the outside, rather than interrogating it from within.

A related distinction, often blurred in the literature, is the one existing between *tactical* and *strategic* use of LLMs. Tactical uses focus on local performance improvements: faster code completion, improved test coverage, reduced effort for isolated development tasks. These contributions are valuable, but they typically leave existing processes and organizational structures largely unchanged. Strategic use, by contrast, presupposes the awareness that software production itself must be reconsidered. It entails the redesign of processes, the adoption of new tools and infrastructures, and the emergence of new competence profiles. Using an LLM occasionally to assist with programming tasks is qualitatively different from engineering an environment in which LLMs operate as continuous cognitive partners embedded within an organization.

It is against this backdrop that the current research landscape must be interpreted. The community has responded to the rapid evolution of LLMs with an impressive volume of manifestos, roadmaps, protocols, taxonomies, and vision papers (Weber 2024; Augusto et al. 2025; Liu et al. 2024). Much of this work is technically sound and has contributed to legitimizing the field and establishing a shared vocabulary. However, it is precisely at this stage of apparent maturity that a structural risk emerges: a growing misalignment between *what is written*, *what is reviewed*, *what is taught*, and *what is actually experienced* when LLMs are used as engineering artefacts over time.

As journal editors, we argue that a significant portion of current research risks privileging narrative coherence over operational exposure, and conceptual positioning over sustained engineering engagement. In this editorial, we examine three manifestations of this risk, namely *(i)* accompanying literature, *(ii)* proxy research, and *(iii)* a nascent review crisis. We then propose concrete steps for the community to address them, leveraging our editorial expertise.

We are aware that an editorial, by its nature, is itself a discursive contribution. We do not exempt ourselves from the tensions we describe. Our argument emerges precisely from recognizing these dynamics in our own practice and in the practices of colleagues whose work we respect. The goal is not to delegitimize conceptual research, but to make visible a structural asymmetry that, if left unexamined, risks distorting the field's trajectory.

2. Accompanying Literature and the Illusion of Progress

A significant portion of current LLM-related research can be described as *accompanying literature*. By this term, we refer to contributions that describe, contextualize, or anticipate the evolution of the field without being directly grounded in sustained, hands-on interaction with LLM-based systems.

Accompanying literature typically focuses on processes rather than their enactment, on frameworks rather than running systems, and on implications rather than constraints. It explains the landscape, often with clarity and rigor, but rarely walks the terrain.

This form of research is not inherently problematic. On the contrary, it is often necessary in the early stages of a technological shift, and some of the most influential contributions to software engineering have been conceptual in nature. But when accompanying literature becomes *dominant*, it creates the illusion of progress, where conceptual refinement substitutes for engineering understanding.

This dynamic is not without precedent in software engineering. The field has already experienced it with the proliferation of mapping studies and systematic literature reviews, which in many cases have become a highly protocolized form of academic production: methodologically rigorous, reproducible, and technically impeccable, yet of limited practical usefulness. The protocols themselves have become the contribution, and the mechanical adherence to them has at times substituted for genuine engagement with the subject matter. The risk with LLM-related research is structurally similar, though amplified by the pace and visibility of the phenomenon, and nurtured by a *publish or perish* frenzy.

A historical analogy is instructive. During the gold rush, the most reliable profits were not made by gold seekers, but by those selling picks, shovels, and wheelbarrows. Today, the analogous risk is the proliferation of conceptual tooling (taxonomies, maturity models, evaluation frameworks) without a proportional increase in knowledge about how LLMs actually behave when used and integrated as part of real software systems.

As editors, we already observe symptoms in the literature, or in submitted papers. A growing number of papers propose "*frameworks for LLM-assisted <XXX>*" in which the framework is the contribution and the LLM interaction with *<XXX>* is illustrative. Benchmarks are constructed and reused without critical examination of whether they reflect real engineering workflows or merely convenient experimental setups. Survey papers synthesize other survey papers. The machinery of academic production operates efficiently, but its coupling to the phenomenon it describes is loosening.

There is a growing risk that LLM-based software engineering research produces more conceptual experts than engineering solutions, privileging narrative authority over engineering outcomes.

3. The Emergence of Proxy Research

Alongside accompanying literature, a second phenomenon has become increasingly visible: *proxy research*. Proxy research is not defined by bad faith, but by distance. Proxy researchers often rely on indirect evidence. They cite datasets produced by others, reuse benchmarks and diagrams they did not construct, and delegate experimentation almost entirely to students or collaborators. Their primary contribution lies in synthesis, narration, and positioning within the research community.

In many cases, their direct interaction with LLMs is limited to sporadic, low-cost, task-level usage: free tiers, short prompts, isolated experiments. As a result, narrative authority begins to exceed experiential authority.

This distance has epistemic consequences. A researcher who has never managed an LLM integration over months of use has not observed prompt decay, the gradual degradation of prompt effectiveness as the underlying model is updated without notice. For example, GPT-4's directly executable code generation dropped from 52% to 10% between March and June 2023, with no version change announced (Chen et al. 2024). This is not a curiosity: it is the kind of silent behavioural drift that breaks production systems and reshapes architectural decisions. Yet it is entirely invisible to a researcher running isolated experiments on a single model snapshot. Similarly, the architectural choice between fine-tuning, retrieval-augmented generation, and increasingly elaborate prompt chains is not an abstract design space; it is a decision forced by cost, latency, and failure rates that only sustained usage reveals. These are not exotic edge cases; they are routine realities of engineering-scale LLM usage, and they are invisible from the vantage point of occasional interaction.

The risk is not that proxy research is incorrect, but that it is incomplete in ways that are invisible to both authors and reviewers. It creates a literature that speaks confidently about systems it has never had to maintain, scale, pay for, or debug. This overconfidence was coined by psychologists (Kruger & Dunning 2000), identifying how hard it is to recognize one's self incompetence when facing a recent phenomenon (and LLMs in software engineering became mainstream very recently).

This is not merely an epistemic limitation; it is an ethical one. Published claims shape how a community understands a technology. Students build on them, practitioners trust them, funding bodies allocate resources on their basis. When that authority rests on indirect or superficial engagement with the phenomenon it describes, the gap between what is claimed and what is known becomes a matter of intellectual responsibility, not just intellectual quality.

We should be clear, however, about what this critique does *not* imply. If genuine understanding of LLM behavior requires sustained, engineering-scale usage costing tens of thousands of dollars per month, then the argument carries an uncomfortable corollary: it risks privileging well-funded laboratories and industry researchers as the only legitimate voices. That would be both unfair and counterproductive. The point is not that every researcher must become an enterprise customer, but that each researcher should be transparent about the vantage point from which they write, and that the community should actively design collaborations, shared infrastructure, and reporting standards that bridge the experiential divide rather than ignore it.

A harder corollary, however, must also be stated. First-order claims about LLM behaviour at engineering scale (drift, decay, organizational failure modes, cost-driven architectural decisions) require engineering-scale exposure. Research operating below that threshold can still produce valuable second-order knowledge: about how the community talks about LLMs, about taxonomies, about pedagogical principles, about regulatory and ethical implications. What it cannot legitimately produce is first-order knowledge about phenomena it has no access to. The way out is not to flatten the asymmetry of authority, but to flatten the asymmetry of access, through industrial residencies, EU-funded shared infrastructure, and consortium agreements with cloud providers. Pretending that this asymmetry does not exist is itself the most consequential form of accompanying literature.

4. A Review Crisis in the Making

A further, more delicate issue concerns the review process itself. Much of the current LLM-related research is written, reviewed, and accepted by scholars who may be excellent software engineers and methodologists, yet who do not themselves operate LLM-centric systems at scale or over time.

This creates a feedback loop in which papers are evaluated primarily on narrative coherence and methodological sophistication, while feasibility, sustainability, and realism are *inferred* rather than experienced. A benchmark evaluation may appear rigorous while testing behaviors that no practitioner would encounter. A framework may seem comprehensive while omitting constraints that any deploying organization would consider fundamental. Over time, this risks shifting the center of gravity of the field away from engineering practice and toward discursive optimization.

This is not a failure of individuals. It is a systemic misalignment between what is being claimed, what is being evaluated, and what is being lived. Diagnosing it from outside the review process would itself be a form of accompanying literature. As editors of this journal, we therefore make the diagnosis actionable. Beginning with this issue, the *Journal of Object Technology* asks authors of LLM-related submissions to include a brief *Engagement Disclosure*: the access tier (free, subscription, API, on-premise), the order of magnitude of interactions and cost over the duration of the work, the model versions involved, and whether the study spans any version transition, context-window change, or pricing change. The disclosure is not a gate. It will not be used to reject papers below a threshold, nor to privilege papers above one. It is an artifact of methodological transparency, comparable to the threats-to-validity statements that empirical software engineering already requires. Once visible, the regime in which a claim was produced becomes a calibration parameter for reviewers and readers, rather than an unstated assumption.

We will also expand reviewer pools for LLM-related submissions to include, where possible, individuals with sustained engineering exposure to LLM-centered systems. We are aware

that this population is currently small, unevenly distributed, and not always reachable through standard recruitment channels. Building it is part of the work.

These are modest measures. They do not flatten the asymmetry of access, and they do not eliminate the risk of a literature whose discursive authority outweighs its operational experience. They do, however, alter the conditions under which that asymmetry remains invisible, which is the precondition for any deeper recalibration this community chooses to pursue.

5. The Economics of Immersion

Using an LLM to perform isolated tasks is not equivalent to building a cognitive infrastructure around LLMs (Mollick 2024; Hutchins 1995). The latter involves sustained usage over extended periods, non-trivial financial cost, prompt evolution and decay, model version drift, latency management, quota constraints, and failure handling. These aspects are not peripheral details. They are central to understanding what it means to *engineer* with LLMs rather than merely experiment with them.

Cost alone is a decisive and underappreciated factor. The economics of LLM usage create a stark experiential divide. Free-tier access to models like GPT-4o or Claude is typically limited to a few dozen queries per day with reduced context windows. A $20/month subscription removes some limits but still constrains volume and model selection. By contrast, API-based usage at engineering scale, processing thousands of files, running regression tests against LLM outputs, and maintaining multi-turn conversations across a codebase, can easily reach $500–$5,000 per month for a single developer, and $20,000–$100,000 per month for an organization.

This economic dimension directly shapes architectural decisions, workflow design, and long-term sustainability. It also shapes what researchers can *observe*. A researcher operating at the free tier inhabits a different phenomenological space than an engineer operating at enterprise scale. The failure modes are different, the optimization pressures are different, and the emergent behaviors are different. Research that does not account for this gradient risks producing results that are formally elegant but operationally irrelevant.

We do not argue that expensive research is inherently better, or that engagement with LLMs should be gated by budget. But we do argue that the *level of immersion* in LLM usage should be made explicit in research contributions. Without such transparency, fundamentally different forms of work become indistinguishable, and the community loses the ability to calibrate claims against experience.

6. An Industry-Led Transformation

Beyond the methodological risks described above, there is a deeper structural concern. The current transformation of software engineering is not being led by academic discourse. It is being led by industrial practice. Companies are redesigning development workflows, renegotiating the boundary between human and automated contributions, and discovering new failure modes at a pace that outstrips the academic publication cycle. Early controlled studies already indicated substantial productivity gains (Peng et al. 2023), and recent enterprise experience reports confirm that these effects compound at organizational scale (Bakal et al. 2025).

This transformation is likely to trigger profound sociotechnical shifts: the emergence of new professional roles, the reshaping or disappearance of existing ones, and a fundamental revision of software development and production processes. Productivity gains enabled by LLM-centered workflows appear to be of an entirely different order of magnitude from previous tool-driven improvements. Such gains will not eliminate the need for human expertise, but they will demand renewed, high-level professional profiles capable of designing, governing, and sustaining LLM-centered engineering infrastructures.

This systemic transformation inevitably extends to education. If software engineering practices, roles, and productivity models are being reshaped at their core, academic curricula cannot remain structurally unchanged. The challenge is not simply to introduce new courses on LLMs, but to rethink which competences are foundational, and whether the traditional progression from programming fundamentals to software design to project management still reflects the actual trajectory of professional formation. If academic research adopts a predominantly accompanying or proxy posture, its capacity to educate professionals for transformed software production environments becomes severely limited. One cannot effectively train practitioners for systemic change while remaining epistemically and operationally distant from it.

7. Where Mitigation Actually Happens

Several of the most cited concerns about LLM-assisted software engineering, including hallucinations, divergence from project conventions, knowledge debt, verification cost, and architectural inconsistency at scale, share a common technical substrate. Their mitigation does not happen at the level of the model, but at the level of how the context window is constructed, maintained, curated, and recovered. Retrieval-augmented architectures, subagent decomposition, plan-then-execute patterns, hierarchical memory structures, prompt caching strategies, and evaluation harnesses for prompt regression form a fast-emerging engineering discipline, with its own design space, its own anti-patterns, and its own empirical lore.

A second, parallel substrate is organizational. Mitigating skill atrophy, redesigning the junior-developer pipeline, restructuring code review around generation rather than authorship, and measuring productivity in conditions where the unit of work has shifted are organizational engineering problems, not purely technical ones. They are being addressed in industrial practice, often artisanally, often by trial and error, and often without explicit articulation.

Both substrates are largely absent from academic LLM-related software engineering research. The result is that mitigation, where it happens, happens through spontaneous and empirical processes whose lessons are neither systematically documented nor scientifically scrutinized. This is precisely the territory in which engineering proximity would generate the

most value: not in proposing yet another framework for "LLM-assisted X", but in studying how teams actually engineer their context, how organizations actually adapt their processes, and which of the practices that survive industrial use deserve to become methodological primitives.

8. Toward Engineering Proximity

The dynamics described in this editorial do not call for abandoning conceptual research or dismissing the contributions of scholars who study LLMs from a primarily analytical perspective. They call for a recalibration, a deliberate effort to increase the proximity between research discourse and engineering reality.

We see several concrete directions that the software engineering community could pursue:

Require transparency on LLM engagement. Research contributions should explicitly report the scale, duration, cost, and modality of LLM interaction underlying their claims. Was the LLM used through a free tier or an enterprise API? Over a single session or across months of development? At what financial cost? Such disclosure would not gatekeep research, but would allow the community to calibrate claims against the depth of underlying experience, much as empirical software engineering already requires transparency about threats to validity.

Operationalize the accompanying/proxy distinction. The distinction between accompanying literature and substantive engagement risks becoming a rhetorical weapon unless it is grounded in observable criteria. We propose four positive indicators that, taken together, distinguish work that has touched the phenomenon from work that has only described it. *Friction*: does the paper report at least one finding that surprised the authors, contradicted prior assumptions, or forced a reconfiguration of the framework? Pure accompanying work has no friction: everything fits. *Intensity of exposure*: not merely which model and tier, but the regime of usage, including order of magnitude of calls, total duration, approximate cost, prompt and response lengths, and presence or absence of extended multi-turn or agentic sessions. Sporadic free-tier use and enterprise API engagement at $5,000 per month are not two levels of the same phenomenon: they are phenomenologically distinct regimes in which drift, decay, and load-correlated failure modes are either visible or invisible *by construction*. The methodological parallel is empirical software engineering, where sample size and duration are reported because they bound what the data can support. *Temporal depth*: does the work span at least one model version transition, context-window change, or pricing change? Treating an LLM as a static object is itself a tell. *Asymmetric finding*: does the paper contain at least one result that pushes back against its own framing? Pure accompanying work confirms its own frame; work that has touched the phenomenon almost always brings home something it did not want. None of these indicators is a gate. Their occasional absence proves nothing. Their systematic absence, however, signals that the paper has been written about other papers rather than about the thing itself.

Invest in longitudinal studies. The most consequential effects of LLM adoption (prompt decay, model drift, organizational adaptation, emerging failure modes) are temporal phenomena. Cross-sectional evaluations, no matter how rigorous, cannot capture them. The community needs longitudinal research designs that follow LLM-integrated development environments over months, not hours.

Build bridges to industrial deployment. Academic-industrial collaboration in this space must go beyond the exchange of datasets and benchmarks. It requires shared exposure to the operational realities of LLM-centered systems: their costs, their failures, their organizational side effects. Residency programs, embedded research collaborations, and joint engineering projects are more likely to produce grounded knowledge than arms-length partnerships.

Rethink educational curricula. Rather than appending LLM modules to existing courses, educational programs should reconsider which competences are foundational in an LLM-augmented landscape. This includes not only technical skills (prompt engineering, model evaluation, infrastructure design) but also critical competences: the ability to assess when LLM outputs are trustworthy, when they are subtly wrong, and when the cost of verification exceeds the benefit of generation.

Some of these directions are already being pursued. Google has published internal accounts of AI-assisted software engineering that foreground engineering constraints and organizational adaptation over isolated capability demonstrations (Chandra 2024). Practitioner-led analyses of LLM-centered development workflows are producing grounded accounts of what works, what fails, and what changes when LLMs are embedded in daily practice (Orosz 2025; Willison 2025). Recent meta-research has begun examining how LLM-related studies report their prompting strategies, offering a first step toward methodological transparency (Korn et al. 2026). These sources are not, of course, without their own systematic distortions (survivorship bias, vendor incentives, hype-cycle pressures, the absence of comparative methodology), and proximity to practice should not be confused with epistemic neutrality. But they document the phenomenon at a regime academic research rarely accesses, and that grounding makes them indispensable interlocutors. These efforts deserve visibility and support; they represent the kind of work that the field most urgently needs.

9. Conclusion

Large Language Models are not merely new tools for Software Engineering. They are becoming engineering media, with economic, cognitive, organizational, and educational consequences. A field that fails to distinguish between tactical uses of LLMs and their strategic integration into software production risks mistaking motion for progress.

At stake is not only the quality of research discourse, but the relevance of industrial practice and the profiles of the professionals being prepared for the coming decade. In the context of a systemic, industry-led transformation, proximity to engineering reality is no longer optional. It is a prerequisite for credible

research, responsible adoption, and meaningful education.

What this requires is not abandoning conceptual research, but acknowledging that engagement at scale is a methodological asset, not a budgetary luxury. The question is no longer whether the field will participate in this transformation, but from what distance.

10. A Note on These Pages

It would be dishonest to close without acknowledging what the reader will likely already have noticed: this editorial is, structurally, part of the literature it critiques. Frameworks about frameworks, manifestos about manifestos, shovels offered to other gold-seekers. We, too, have set up a stall.

Our defence is modest. We have tried to write from inside the trap rather than above it, and to make visible the asymmetry that pages like these too often leave implicit. If this is itself a pickaxe, we hope it is at least one that suggests where *not* to dig. The verdict, as with any verdict on an LLM-related paper today, should arrive after the reader has logged into their enterprise console, not before.

References

Augusto, C., Bertolino, A., De Angelis, G., Lonetti, F., & Morán, J. (2025). Large language models for software testing: A research roadmap. *arXiv preprint arXiv:2509.25043*.

Bakal, G., Dasdan, A., Katz, Y., Kaufman, M., & Levin, G. (2025). Experience with github copilot for developer productivity at zoominfo. *arXiv preprint arXiv:2501.13282*.

Chandra, S. (2024). Ai in software engineering at google: Progress and the path ahead (invited talk). In *Proceedings of the 1st acm international conference on ai-powered software* (pp. 182–182).

Chen, L., Zaharia, M., & Zou, J. (2024). How is ChatGPT's behavior changing over time? *Harvard Data Science Review*. (Originally arXiv:2307.09009 (2023). Data and prompts at https://github.com/lchen001/LLMDrift)

Chu, X., Talluri, S., Lu, Q., & Iosup, A. (2025). An empirical characterization of outages and incidents in public services for large language models. In *Proceedings of the 16th acm/spec international conference on performance engineering (icpe '25)*. Toronto, ON, Canada. (arXiv:2501.12469. Data and code at https://github.com/atlarge-research/llm-service-analysis)

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., … Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*. doi: 10.1145/3695988

Hutchins, E. (1995). *Cognition in the wild*. MIT Press.

Korn, A., Zaruchas, L., Arora, C., Metzger, A., Smolka, S., Wang, F., & Vogelsang, A. (2026). Reporting llm prompting in automated software engineering: A guideline based on current practices and expectations. *arXiv preprint arXiv:2601.01954*.

Kruger, J., & Dunning, D. (2000, 01). Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, *77*, 1121-34. doi: 10.1037//0022-3514.77.6.1121

Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., & Lou, Y. (2024). Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977*.

Mollick, E. (2024). *Co-intelligence: Living and working with AI*. Portfolio/Penguin.

Orosz, G. (2025). *Software engineering with LLMs in 2025: Reality check*. https://newsletter.pragmaticengineer.com/p/software-engineering-with-llms-in-2025.

Pan, G., Chodnekar, V., Roy, A., & Wang, H. (2025). A cost-benefit analysis of on-premise large language model deployment: Breaking even with commercial LLM services. *arXiv preprint arXiv:2509.18101*.

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*.

Weber, I. (2024). Large language models as software components: A taxonomy for LLM-integrated applications. *arXiv preprint arXiv:2406.10300*.

Willison, S. (2025). *2025: The year in LLMs*. https://simonwillison.net/2025/Dec/31/the-year-in-llms/.

Zhang, Q., Fang, C., Xie, Y., Zhang, Y., Yang, Y., Sun, W., … Chen, Z. (2025). A survey on large language models for software engineering. *Science China Information Sciences*. (Also available as arXiv:2312.15223)

About the authors

Önder Babur is assistant professor at Wageningen University & Research (The Netherlands). His research interests include model-driven engineering, model analytics, and machine learning for software engineering.

Sébastien Mosser is associate professor of software engineering at McMaster University (Canada). His research interests include scalable software composition, domain-specific languages, and modelling applied to cloud computing, cyber-physical systems, and micro-service architectures.

Alfonso Pierantonio is professor at the Università degli Studi dell'Aquila (Italy). His research interests include software engineering, model-driven engineering, and the socio-technical impact of software technologies.