International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Large Language Models for Software Engineering: A Review of Methods, Evaluation Frameworks, and Practical Deployment Considerations

Sead Mašetić[1]; Dino Kečo[2]; Fikret Zajmović[3]

[1, 2, 3]Department of Information Technologies, International Burch University, Bosnia and Herzegovina
[1] sead.masetic@stu.ibu.edu.ba; [2] dino.keco@ibu.edu.ba; [3] fikret.zajmovic@stu.ibu.edu.ba

DOI: https://doi.org/10.47760/ijcsmc.2026.v15i11.001

Abstract: Large language models have become a practical component of software development, with AI-powered coding assistants now integrated into real workflows across the industry. Research in this area has advanced rapidly, producing models capable of generating, editing, and reasoning about code at a level that demonstrably improves developer productivity. Progress has been uneven, however. Benchmark performance has improved consistently while the gap between evaluation results and real-world applicability has remained wide — particularly for tasks requiring repository-level reasoning, long-context understanding, and reliable output under realistic conditions. Less examined is the practical dimension of deployment. Most research assumes cloud-based infrastructure, leaving questions around privacy, operational cost, hardware constraints, and the viability of locally deployed models largely unaddressed. This review examines the current state of large language models for software engineering through three organizing questions: how code LLMs have evolved in capability and design philosophy; what current evaluations actually measure and what they miss; and what practical barriers to deployment exist, and how the choice between local and cloud-based systems changes the picture. The review covers foundation models for code, evaluation benchmarks, repository-level reasoning approaches, agent-based systems, and practical deployment limitations including hallucinations, security vulnerabilities, and privacy risks. A consistent pattern emerges across the literature: the field has optimized for measurable performance while leaving practical deployment questions underexplored. The absence of systematic evaluation comparing local and cloud-based coding assistants is identified as a concrete open research direction. This includes the need for deployment-aware benchmarks and empirical investigation of whether a hybrid workflow — local models for execution, cloud models for planning — can reduce cloud dependency without sacrificing output quality.

Keywords: large language models, software engineering, code generation, local deployment, AI coding assistants, benchmarking, evaluation frameworks

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

I. INTRODUCTION

Large language models have become a practical part of software development. Tools like GitHub Copilot [1], Claude Code [2], and Gemini CLI [3] are now used in real workflows, not just research settings, and studies confirm that developers using them complete tasks faster and with less effort than those working without them [4]. The question is no longer whether these systems work — it is whether the conditions under which they work are acceptable for everyone who might use them.

Most AI coding assistants currently operate as cloud-based services. This means sending code to external servers, paying recurring subscription or usage fees, and depending on a stable internet connection. For individual developers or small teams these constraints are inconvenient. For organizations handling sensitive or proprietary codebases they can be prohibitive. These are not edge cases — they describe a large portion of real development contexts.

Locally deployed open-source models offer a different approach. Qwen2.5-Coder [5] and DeepSeek-Coder [6] run on consumer hardware without sending code anywhere. That changes the privacy and cost equation entirely. Existing benchmark evaluations of open-source models [7], [8] do not account for these deployment differences, and how local models actually compare to cloud-based tools under realistic conditions has not been addressed in the literature reviewed here.

One specific gap stands out. Existing evaluations treat software engineering tasks as a single category, measuring overall performance without distinguishing between tasks that require high-level planning and reasoning versus tasks focused on concrete implementation. These may place very different demands on a model. It is plausible that local models, which operate under greater hardware constraints, perform competitively on execution-oriented tasks such as code generation and bug fixing, even if they fall short on planning-heavy scenarios. If that is the case, a hybrid workflow — cloud models for planning, local models for execution — could reduce cloud dependency without meaningfully sacrificing output quality. This possibility has not been a focus of the literature reviewed here.

This review addresses these gaps by examining the current state of large language models for software engineering through three organizing questions. First, how have code LLMs evolved in capability and design philosophy? Second, what do current evaluations actually measure — and what do they miss? Third, what are the practical barriers to deployment, and how does the choice between local and cloud-based systems change the picture? These questions structure the review and connect its individual sections into a coherent argument rather than a catalog of findings.

The review covers foundation models for code, evaluation benchmarks, repository-level reasoning approaches, agent-based systems, and practical deployment limitations including hallucinations, security vulnerabilities, and privacy risks. It does not cover LLM training infrastructure, formal verification methods, or non-English code contexts. A reader finishing this review should have a clear understanding of what the field has learned, where the evidence is strong, where assumptions have gone unexamined, and what empirical work is most needed.

II. METHODOLOGY

This review is conducted as a structured literature review. Unlike a comprehensive systematic review, it does not aim for exhaustive coverage of every published paper in the field. Instead, it applies transparent selection criteria and a defined classification framework to identify and synthesize the most directly relevant work across five thematic areas. The methodology is reported in sufficient detail to allow readers to evaluate the basis for the review's findings and the scope of its coverage.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

A. Research Questions

Three research questions structure this review. They derive from the gaps identified in the introduction and organize the presentation of findings across all subsequent sections.

- **RQ1** — How have code-oriented large language models evolved in capability, architecture, and design philosophy between 2020 and 2025?
- **RQ2** — What do current evaluation frameworks and benchmarks actually measure, and what systematic limitations do they exhibit?
- **RQ3** — What practical barriers exist to deploying LLMs for software engineering tasks, and how does the choice between local and cloud-based infrastructure affect them?

B. Search Strategy and Scope

Searches were conducted across academic databases covering peer-reviewed journals, conference proceedings, and arXiv preprints relevant to the field. The search covered publications from January 2020 to March 2025, a window that captures the transition from early code-specific pre-training approaches to the current generation of efficient, locally deployable models.

Seventeen conceptual search terms were identified across five thematic clusters corresponding to the review's research questions. These terms were selected to surface papers thematically relevant to the review's scope, rather than to locate specific known papers. Each search was conducted with a date filter applied. For searches returning more than 500 results, the top 30 records sorted by relevance were examined, as relevance-ranked results reliably surface the most cited and directly pertinent work within the first results for specific multi-term queries. Table I presents the search terms used and the rationale for their inclusion.

TABLE I
SEARCH TERM CLUSTERS AND RATIONALE

Category	Search Term	Rationale
Foundation Models & Architecture	code generation LLM	Core term for identifying papers on generative code models
	code pre-training	Targets architecture and training methodology papers
	open-source code model deployment	Surfaces papers on accessible, locally deployable models
Benchmarks & Evaluation	repository-level evaluation	Identifies benchmark papers beyond single-function tasks
	code evaluation framework	Broader term for evaluation methodology papers
Repository Reasoning	repository-level code completion	Directly targets cross-file context and completion papers
	retrieval augmented code generation	Identifies RAG-based approaches for code tasks
Agent Systems	coding agent	Core term for LLM-based autonomous coding systems
	tool-use LLM software engineering	Surfaces papers on tool-integrated agent workflows
	autonomous coding	Captures self-directed code generation and repair systems
	self-repair code	Targets iterative self-improvement and refinement approaches

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Category	Search Term	Rationale
Deployment & Practical Barriers	local LLM deployment	Primary term for on-device and offline deployment papers
	on-premise code assistant	Targets enterprise and privacy-sensitive deployment contexts
	privacy LLM code	Surfaces papers on data privacy risks in AI coding tools
	LLM hallucination code	Identifies reliability and hallucination studies in code generation
	LLM generated code security	Targets vulnerability and security studies in generated code
	AI coding assistant developer productivity	Captures empirical adoption and productivity studies

In addition to the keyword searches, forward and backward citation tracking was applied to papers retrieved through the conceptual searches. The backward method involved reviewing the reference lists of selected papers to identify foundational works cited across multiple studies. The forward method involved identifying papers that cited the selected works. This step was the primary mechanism through which specific benchmark papers, model papers, and empirical studies central to the field — such as those introducing HumanEval, SWE-bench, RepoBench, and RepoQA — were formally identified and included. Papers accessed through citation tracking were subject to the same inclusion and exclusion criteria as those retrieved through keyword searches.

C. Inclusion and Exclusion Criteria

All candidate papers were assessed against a defined set of inclusion and exclusion criteria, applied consistently during abstract screening and full-text review. The criteria are summarised in Table II.

TABLE II
INCLUSION AND EXCLUSION CRITERIA

Criterion	Inclusion	Exclusion
Time Period	2020–2025	Before 2020
Relevance	Directly addresses at least one of the three research questions	NLP papers without direct code application; training infrastructure without SE relevance
Publication Type	Peer-reviewed papers, workshop papers, arXiv preprints with significant citation counts	Blog posts, grey literature, vendor white papers, non-peer-reviewed technical reports
Language	English-language publications	Non-English publications
Scope	Code generation, evaluation frameworks, agent systems, or LLM deployment	Formal verification without LLMs; non-English code contexts; edge computing and mobile inference

The rationale for each criterion is as follows. The 2020–2025 time frame reflects the period of most consequential development in LLMs for software engineering, capturing the introduction of Codex (2021) through to the most recent generation of locally deployable models (2024–2025). The scope constraint — requiring direct relevance to at least one of the three research questions — ensures the review does not stray into adjacent topics such as LLM training infrastructure or formal verification methods. Restricting to English-language publications is acknowledged as a limitation in the *Limitations* section of this paper.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

D. Paper Selection Process

The 17 keyword searches, each capped at 30 results where applicable, yielded a total of 468 records. Following the removal of duplicate records identified across multiple search terms — particularly between thematically adjacent clusters such as agent systems and repository reasoning — 271 unique records remained for abstract-level screening.

Abstract screening against the inclusion and exclusion criteria resulted in the exclusion of 208 records. The most common reasons for exclusion were: papers addressing natural language processing without direct application to code (n=92); papers focused exclusively on LLM training infrastructure without relevance to software engineering evaluation or deployment (n=71); and papers published outside the 2020–2025 review window (n=45). The remaining 63 papers underwent full-text review.

At the full-text stage, a further 34 papers were excluded: 22 for addressing topics explicitly out of scope (formal verification, non-English code contexts, mobile inference); and 12 for insufficient methodological transparency to meet the quality assessment threshold described in Section II-F. Of the 29 papers remaining, 7 were excluded during quality assessment, leaving 22 papers from the keyword search stream. Papers identified through citation tracking — 8 in total — were added at the full-text stage, all of which met the inclusion criteria and passed quality assessment. The final review is based on 30 papers. Fig. 1 illustrates the complete selection process.

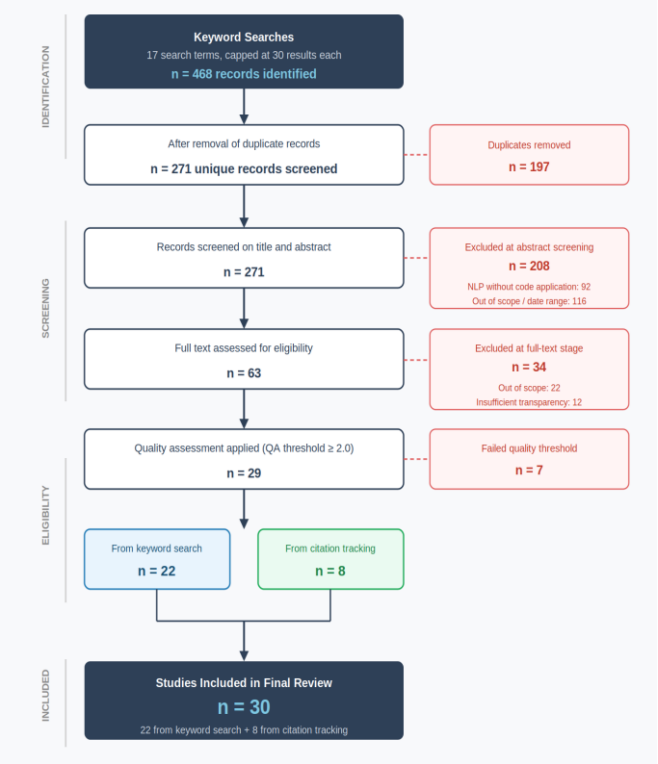


Fig. 1 Paper Selection and Screening Process

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

E. Classification Framework

To ensure structured and transparent analysis, each included paper was classified along three dimensions. This framework provides the organizing structure for the Results section and enables systematic comparison across the literature.

1) Research Category: Papers were assigned to one of four primary categories based on their principal contribution:

- Foundation Models and Architecture — Papers introducing or evaluating specific model architectures or training approaches for code-oriented LLMs (e.g., Codex, Code Llama, Qwen2.5-Coder, DeepSeek-Coder).
- Benchmarks and Evaluation — Papers proposing or analyzing evaluation frameworks, metrics, or benchmarking methodologies for code generation and software engineering tasks (e.g., HumanEval, SWE-bench, RepoBench, RepoQA).
- Agent Systems and Repository Reasoning — Papers addressing multi-step, tool-integrated systems or retrieval-augmented approaches for repository-level tasks (e.g., SWE-agent, Reflexion).
- Practical Barriers and Deployment — Papers examining limitations of LLMs in real-world use, including hallucinations, security vulnerabilities, privacy risks, and deployment considerations (e.g., CodeCloak, CodeMirage, SecRepoBench).

2) Methodology Type: Papers were additionally classified by their primary research methodology:

- System or Architecture Paper — Introduces a new model, tool, or system and presents its design and benchmark performance.
- Empirical Study — Measures real-world behavior under naturalistic conditions, including user studies and field evaluations.
- Benchmark Paper — Proposes a new evaluation dataset, task set, or testing framework.
- Survey or Review — Synthesizes and evaluates existing literature across a topic area.

3) Deployment Context: Given this review's central focus on the practical distinction between local and cloud-based deployment, each paper was classified by the infrastructure it assumes or explicitly evaluates:

- Cloud-Dependent — The system or evaluation assumes cloud-based infrastructure and could not be reproduced on consumer-grade hardware without significant modification.
- Deployment-Agnostic — The paper does not specify deployment context, or its findings apply regardless of infrastructure choice.
- Local-Capable — The system or evaluation explicitly supports or targets local deployment on consumer hardware, without requiring cloud services.

F. Quality Assessment

All 30 papers included in the final review were assessed against four quality criteria before final inclusion decisions were made. These criteria are adapted from established guidelines for systematic and narrative literature reviews [9], [10]. Table III presents the scoring scheme.

TABLE III
QUALITY ASSESSMENT CRITERIA AND SCORING SCHEME

Quality Criterion	Fully Met	Partially Met / Not Met
QA1: Research objective clearly stated	1.0	0.5 / 0.0
QA2: Methodology described in sufficient detail	1.0	0.5 / 0.0

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Quality Criterion	Fully Met	Partially Met / Not Met
QA3: Findings supported by evidence (results, data, measurements)	1.0	0.5 / 0.0
QA4: Limitations of the work acknowledged	1.0	0.5 / 0.0
Maximum score	4.0	Threshold for inclusion: ≥ 2.0

Papers scoring below 2.0 were excluded from the review. This threshold was applied as a binary filter — a minimum standard of methodological transparency — rather than as a quality ranking that influenced how findings were weighted in the narrative synthesis. Seven papers were excluded at this stage. The final review includes 30 papers: 22 from the keyword search stream and 8 identified through citation tracking, all of which meet the minimum quality threshold.

III. Evolution of Code-Oriented Language Models

A. Early Representation Models

Early work on language models for code was not about generation — it was about understanding. CodeBERT [11] introduced joint pre-training over natural language and programming language, making tasks like code search and documentation generation more tractable. It worked reasonably well, but the token sequence representation was a fundamental constraint. Relationships between variables, control flow, and dependencies simply do not survive tokenization intact. GraphCodeBERT [12] incorporated data flow graphs directly into pre-training to address this. The results improved, which was not surprising — if you give a model explicit information about how data moves through code, it understands code better.

B. The Shift to Code Generation

Generation became the dominant focus with Codex [7]. Fine-tuned on a large corpus of public code, it could produce executable programs from natural language descriptions well enough to be practically useful. It also introduced HumanEval [7], shifting evaluation away from surface-level similarity metrics toward functional correctness measured by unit tests. That was a meaningful methodological change. The benchmark results, however, told only part of the story. Codex struggled with complex reasoning tasks, made variable binding errors, and produced logically inconsistent code more often than the aggregate scores suggested. Generating a correct solution frequently required multiple attempts, which raised questions about how reliable the model actually was in practice. A system that passes benchmarks on average but fails unpredictably is a different proposition from one that is consistently dependable.

C. Scale, Data Quality, and Instruction Tuning

CodeGen [13] scaled open-source code generation, making powerful models publicly available for the first time at this level of capability. InCoder [14] shifted focus to infilling — inserting code at arbitrary positions within existing files using bidirectional context. Most development work involves editing existing code rather than writing new code from nothing. Both models made genuine progress in their respective areas, but cross-file reasoning and long-range context remained outside their reach.

Code Llama [15] took an existing general-purpose LLM and adapted it for code through longer context windows and instruction tuning. The benchmark results on HumanEval, MBPP [16], and MultiPL-E [17] were strong enough to suggest this was a reasonable shortcut compared to building a code-specific architecture from scratch. StarCoder2 [18] came at the problem differently. Rather than scaling up parameters, it invested in data — The Stack v2 covers hundreds of programming languages and was curated more carefully than most training corpora at the time. The resulting model outperformed larger ones trained on noisier data, which made a

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

reasonable case that what you train on matters as much as how big your model is. CodeT5+ [19] unified generation, understanding, and retrieval in one model rather than maintaining separate systems for each task.

D. Locally Deployable Open-Source Models

The Qwen2.5-Coder family [5] spans 1.5B to 72B parameters, with smaller variants achieving competitive benchmark results through data quality rather than scale alone. DeepSeek-Coder [6] is built around the same tradeoff — a smaller, more efficient model that does not sacrifice generation quality in the process. Neither family tops the leaderboards, but that is not really the point. The value is not that they beat cloud models — it is that they do not require them.

Looking across this trajectory, a pattern emerges that individual papers do not always make explicit. The field moved through three distinct phases: an initial focus on understanding code structure, a shift toward generation at scale, and a more recent turn toward efficiency and accessibility. Each phase addressed limitations exposed by the previous one. Representation models could not generate. Generation models were unreliable and resource-intensive. Scale-focused models improved reliability but remained out of reach for most local deployment scenarios.

The efficiency turn — exemplified by Qwen2.5-Coder and DeepSeek-Coder — is not simply a continuation of the scaling trend. It reflects a different design philosophy: that capable code assistance does not require unlimited infrastructure. This shift has practical consequences that the research literature has not yet fully examined. Models now exist that are small enough to run on consumer hardware and capable enough to be useful. What the field has not yet established is precisely where their usefulness ends and where cloud-based systems remain necessary — and that boundary may be task-dependent rather than absolute.

IV. EVALUATION FRAMEWORKS AND BENCHMARKS

A. Early Standardization Efforts

How progress is measured shapes what progress looks like. Early benchmarks like CodeXGLUE [20] standardized evaluation across a range of code-related tasks — understanding, translation, summarization — and made cross-model comparison more systematic. That was useful, but the tasks were small and self-contained. They did not reflect how code actually exists in real projects, spread across multiple files with interdependencies that matter.

B. Functional Correctness Evaluation

HumanEval shifted the conversation by measuring functional correctness rather than surface similarity. Instead of checking whether generated code looked like a reference solution, it checked whether the code actually worked — using executable unit tests and pass@k as the primary metric. This was a genuine improvement. The limitation is that HumanEval problems are isolated, single-function tasks. Solving them well says something about a model's generation capability, but not much about whether it can operate inside a real codebase. MBPP [16] broadened the pool of Python problems and MultiPL-E [17] extended the same approach across multiple programming languages. Both were useful expansions. Neither escaped the core problem — every task still lives in isolation, disconnected from the dependencies, context, and complexity that characterize real software projects.

C. Repository-Level Evaluation

RepoBench [21] and RepoQA [22] moved evaluation into repository territory. RepoBench built multi-file completion tasks where models had to pull relevant information from across a codebase rather than a single file. RepoQA took a different angle, testing whether models could locate relevant functions within large repositories

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

given a natural language query. RepoBench found that completion performance drops substantially when tasks cross single-file boundaries. RepoQA revealed a related problem — models struggle to accurately retrieve and apply information distributed across extended repository structures.

SWE-bench [8] is the closest thing to a realistic evaluation framework the field currently has. Tasks come from actual GitHub issues — models have to read a problem description, find their way around a codebase, make the right changes, and pass the associated tests. That covers the full development cycle, not just one isolated step. The results are not encouraging. Even the strongest available models resolve only a small fraction of tasks, which puts a ceiling on how much can be read into performance on simpler benchmarks. Chen et al. [23] make a related point — evaluation across the field remains fragmented, with no single metric able to capture correctness, efficiency, readability, and usability at the same time.

D. What Current Benchmarks Do Not Measure

Despite this progression, a consistent blind spot remains. Evaluation focuses almost entirely on output quality — whether the generated code is correct, complete, and functional. What is not measured is the conditions under which that output is produced. Deployment environment, hardware constraints, latency, operational cost, and data privacy are absent from every major evaluation framework reviewed here. Two systems producing identical output quality may differ substantially in cost, speed, and the privacy guarantees they offer — yet existing benchmarks treat them as equivalent.

This omission reflects a shared assumption embedded in how benchmarks are designed: that infrastructure is unlimited and deployment conditions are outside the scope of evaluation. That assumption holds reasonably well when comparing cloud-based systems against each other. It breaks down entirely when the question is whether a locally deployed model is viable for a given task. As locally deployable models become increasingly capable, evaluation frameworks that ignore deployment conditions become increasingly inadequate tools for answering the questions practitioners actually care about.

E. Benchmark Comparison Table

The table below summarizes the major benchmarks reviewed, their scope, primary metrics, and key limitations. The Deployment Aware column indicates whether deployment conditions are accounted for in the evaluation design.

TABLE IV
BENCHMARKS COMPARISON

Benchmark	Task Type	Scope	Primary Metric	Deployment Aware	Key Limitation
CodeXGLUE [20]	Understanding, translation, summarization	Single file	BLEU, Accuracy, F1, CodeBLEU, MRR	No	Tasks are small and isolated, do not reflect real project complexity
HumanEval [7]	Code generation	Single function	pass@k	No	Isolated problems, no repository context
MBPP [16]	Code generation	Single function	pass@k	No	Limited to Python, isolated tasks
MultiPL-E [17]	Code generation	Single function	pass@k	No	Extends HumanEval across languages but retains isolation

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Benchmark	Task Type	Scope	Primary Metric	Deployment Aware	Key Limitation
RepoBench [21]	Code completion	Multi-file repository	Exact match, edit similarity, Accuracy@k	No	Does not cover full engineering cycle
RepoQA [22]	Code retrieval	Large repository	Retrieval accuracy (%)	No	Retrieval only, no generation or fixing
SWE-bench [8]	Bug fixing, navigation	Full repository	% resolved	No	Requires significant infrastructure, no deployment constraints

V. CODING AGENTS AND TOOL USE

A. *Tool-Integrated Agent Systems*

Standalone model performance has a ceiling. Even with better training data and larger context windows, a model that generates code in a single pass cannot navigate a repository, run tests, or respond to execution feedback. Agent-based systems address this by giving models access to tools and environments, turning code generation into an interactive process rather than a one-shot prediction.

SWE-agent [24] is a clear example. Rather than generating a solution directly, the model navigates a repository, edits files, and runs tests iteratively until a solution passes. SWE-bench [8] scores improved considerably over standalone models handling the same tasks. The models involved were not fundamentally more capable — they just had the ability to try something, see what happened, and change course accordingly.

B. *Iterative Self-Improvement*

Reflexion [25] turned this idea inward rather than outward. Instead of acting on an external environment, the model examines its own previous outputs, stores what it learns as text, and uses that to do better on the next attempt. No retraining required. It worked reasonably well on coding tasks, but there is an obvious weak point — the whole thing depends on the model being able to accurately identify what went wrong. That is not a given.

C. *Empirical Evidence from Real-World Use*

There is reasonable empirical support for the usefulness of these tools in practice. Peng et al. [4] measured productivity gains among GitHub Copilot users — tasks got done faster, with less effort. Sergeyuk et al. [26] found adoption concentrated around repetitive tasks rather than complex ones, which is telling in itself. Both studies also surfaced the same complaints: inaccurate outputs, limited contextual awareness, and a trust gap that developers have not fully closed.

D. *The Unstated Infrastructure Assumption*

What the agent-based systems reviewed here share is an assumption that rarely gets examined explicitly. SWE-agent, Reflexion, and the tools evaluated in the productivity studies were all built for and tested on cloud-based infrastructure. The interaction loops, tool integrations, and iterative workflows that make these systems effective depend on fast, reliable inference — something cloud deployment provides by design.

This is not a neutral engineering choice. It is a methodological constraint that shapes what these systems can do, who can use them, and what counts as a viable result. A developer running a locally deployed model on consumer hardware faces a fundamentally different set of constraints: slower inference, limited memory, and no access to the cloud-side tooling that agent benchmarks assume. Whether agent-based workflows are viable

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

under these conditions has not been tested in the literature reviewed here — and the gap runs deeper than hardware. It runs through how agent research is framed — what counts as a valid test environment, what infrastructure is assumed, and whose deployment conditions are treated as the default. Those choices have consequences that the field has not yet examined directly.

VI. Practical Barriers to Deployment

Benchmark scores and agent capabilities only matter if the tools can actually be used. The limitations covered in this section are not edge cases or minor inconveniences — they are the reasons developers hesitate, organizations hold back, and adoption remains more cautious than the performance numbers would suggest.

A. Hallucinations and Reliability

Hallucination is the most pervasive. The problem with hallucinated code is not that it looks wrong — it is that it looks right. Syntactically valid, plausibly structured, and quietly incorrect. CodeMirage [27] found that these failures are hard to catch for exactly that reason. A developer scanning the output has no obvious signal that the API being called does not exist or that the logic will break under certain conditions.

B. Security Vulnerabilities

Security compounds this. Mohsin et al. [28] found that code generated by LLMs can contain security vulnerabilities even when it passes functional tests — working code and safe code are not the same thing. SecRepoBench [29] took this further by testing at the repository level, finding that the correctness-security gap does not disappear when tasks get more realistic. It gets harder to close.

C. Trust Gaps in Practice

The accuracy and hallucination problems feed directly into a trust problem. Sergeyuk et al. [26] found that developers have not fully bought into AI coding assistants — outputs get reviewed, double-checked, and sometimes discarded. Given what the reliability studies show, that skepticism makes sense. The issue is that constant verification narrows the gap between using AI assistance and not using it.

D. Privacy and the Case for Local Deployment

Privacy is where the practical stakes are highest, and where the choice of deployment model matters most directly. Cloud-based coding assistants require code to be transmitted to external servers as a condition of use. For many development contexts this is simply not acceptable — not because of preference, but because of regulatory requirements, contractual obligations, or the competitive sensitivity of the codebase.

CodeCloak [30] made the data leakage risk concrete. Rather than accepting the exposure risk, it obfuscates code before transmission, preserving model utility while reducing exposure. The existence of such a tool illustrates that the privacy risk is real enough to require dedicated engineering solutions. For enterprise users the question is not about trusting the service provider — it is about whether transmitting code externally is acceptable at all.

Local deployment substantially reduces this risk by design — code remains within the local environment and is not transmitted to external servers. This is not a minor convenience — for a substantial portion of real development contexts, it is the difference between a tool that can be adopted and one that cannot, regardless of its performance on benchmarks.

Taken together, the limitations reviewed in this section point toward something the field has been slow to acknowledge: that the barriers to adoption are not purely technical. Hallucinations can be partially mitigated through review practices. Security vulnerabilities can be caught through static analysis. Trust gaps narrow as models improve. Privacy risks, however, cannot be engineered away within the cloud deployment model. They

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

are architectural. And that makes local deployment not just a cost-saving alternative but a structural necessity for a meaningful portion of potential users.

VII. RESULTS

This section presents the findings of the review, organized first by a descriptive overview of the literature and then by each of the three research questions. The classification framework described in Section II-E provides the structure for the overview. Detailed analysis of individual papers and thematic areas is carried out in the thematic sections that follow.

A. Overview of Reviewed Literature

The 30 papers included in this review span January 2020 to March 2025. Publication volume increased notably from 2022 onward, with 2023 and 2024 together accounting for 60% of the final sample. This trend reflects the rapid growth of the field following the public release of Codex in 2021 and the subsequent proliferation of open-source models and evaluation frameworks. Table V summarises the distribution of included papers across publication year, venue type, research category, and deployment context.

TABLE V
DISTRIBUTION OF REVIEWED PAPERS BY YEAR, VENUE, RESEARCH CATEGORY, AND DEPLOYMENT CONTEXT

Dimension	Category	Count (%)
Publication Year	2020	1 (3%)
	2021	4 (13%)
	2022	3 (10%)
	2023	8 (27%)
	2024	10 (33%)
	2025	4 (13%)
Venue Type	Top-tier conference (NeurIPS, ICLR, EMNLP, ACL, NAACL)	8 (27%)
	Peer-reviewed journal	1 (3%)
	arXiv preprint	21 (70%)
Research Category	Foundation Models and Architecture	10 (33%)
	Benchmarks and Evaluation	9 (30%)
	Agent Systems and Repository Reasoning	5 (17%)
	Practical Barriers and Deployment	6 (20%)
Deployment Context	Cloud-Dependent	15 (50%)
	Deployment-Agnostic	10 (33%)
	Local-Capable	5 (17%)

The venue distribution is notable. Only 27% of reviewed papers were published at top-tier peer-reviewed conferences, and just one paper appeared in a peer-reviewed journal. The remaining 70% exist as arXiv preprints. This reflects the pace of the field: in LLM research, the publication cycle of traditional venues is too slow to keep pace with model releases, and arXiv has become the de facto primary venue for dissemination. This distribution is itself a methodological observation — the reviewed literature is predominantly non-peer-reviewed in the traditional sense, which is why the quality assessment criteria applied in Section II-F are particularly important.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

The research category distribution shows that Foundation Models and Benchmarks together account for 63% of reviewed papers, confirming that the literature has focused heavily on building and evaluating systems rather than on studying deployment conditions or adoption barriers. Practical Barriers and Agent Systems together represent 37% — a meaningful share, but one that underscores the gap between primary research focus and the deployment realities faced by practitioners.

The deployment context distribution is the most consequential finding of the overview. Only 17% of reviewed papers explicitly support or evaluate local deployment on consumer hardware. The remaining 83% either assume cloud infrastructure (50%) or do not address deployment context at all (33%). This asymmetry is itself a finding: the privacy and cost concerns most frequently cited as adoption barriers are precisely the concerns that local deployment addresses, yet local deployment has received comparatively limited systematic research attention.

Table VI provides the full list of reviewed papers with year, venue, and research category.

TABLE VI
REVIEWED PAPERS WITH PUBLICATION METADATA

Paper	Year	Venue	Research Category
Feng et al. [11]	2020	EMNLP (Findings)	Foundation Models
Chen et al. [7]	2021	arXiv	Foundation Models / Benchmarks
Guo et al. [12]	2021	ICLR	Foundation Models
Lu et al. [20]	2021	arXiv	Benchmarks
Austin et al. [16]	2021	arXiv	Benchmarks
Fried et al. [14]	2022	arXiv	Foundation Models
Nijkamp et al. [13]	2022	arXiv	Foundation Models
Cassano et al. [17]	2022	arXiv	Benchmarks
Wang et al. [19]	2023	EMNLP	Foundation Models
Zhang et al. [31]	2023	EMNLP	Agent / Repository
Peng et al. [4]	2023	arXiv	Practical Barriers
Shinn et al. [25]	2023	NeurIPS	Agent / Repository
Liu et al. [21]	2023	arXiv	Benchmarks
Rozière et al. [15]	2023	arXiv	Foundation Models
Jimenez et al. [8]	2023	arXiv	Benchmarks
Shrivastava et al. [32]	2023	arXiv	Agent / Repository
Lozhkov et al. [18]	2024	arXiv	Foundation Models
Hui et al. [5]	2024	arXiv	Foundation Models
Guo et al. [6]	2024	arXiv	Foundation Models
Liu et al. [22]	2024	arXiv	Benchmarks
Agarwal et al. [27]	2024	arXiv	Practical Barriers
Mohsin et al. [28]	2024	arXiv	Practical Barriers
Finkman Noah et al. [30]	2024	arXiv	Practical Barriers
Chen et al. [23]	2024	arXiv	Benchmarks
Sergeyuk et al. [26]	2024	J. Systems & Software	Practical Barriers
Yang et al. [24]	2024	NeurIPS	Agent / Repository
Chen et al. [33]	2025	arXiv	Benchmarks

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Paper	Year	Venue	Research Category
Li et al. [34]	2025	ACL	Benchmarks
Liu et al. [35]	2025	NAACL	Agent / Repository
Shen et al. [29]	2025	arXiv	Practical Barriers

B. Classification of Reviewed Papers

Table VII provides the classification of all 30 reviewed papers across the three dimensions of the framework described in Section II-E. The analysis reveals patterns that are discussed in detail in each research question section below.

TABLE VII
CLASSIFICATION OF REVIEWED PAPERS BY RESEARCH CATEGORY, METHODOLOGY TYPE, AND DEPLOYMENT CONTEXT

Paper	Research Category	Methodology	Deployment Context
Feng et al. [11] — CodeBERT	Foundation Models	System / Architecture	Cloud-Dependent
Guo et al. [12] — GraphCodeBERT	Foundation Models	System / Architecture	Cloud-Dependent
Chen et al. [7]	Foundation Models / Benchmarks	System / Architecture	Cloud-Dependent
Lu et al. [20] — CodeXGLUE	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Austin et al. [16] — MBPP	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Nijkamp et al. [13] — CodeGen	Foundation Models	System / Architecture	Cloud-Dependent
Fried et al. [14] — InCoder	Foundation Models	System / Architecture	Cloud-Dependent
Cassano et al. [17] — MultiPL-E	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Wang et al. [19] — CodeT5+	Foundation Models	System / Architecture	Local-Capable
Zhang et al. [31] — RepoCoder	Agent Systems	System / Architecture	Cloud-Dependent
Jimenez et al. [8] — SWE-bench	Benchmarks & Evaluation	Benchmark Paper	Cloud-Dependent
Peng et al. [4] — GitHub Copilot study	Practical Barriers	Empirical Study	Cloud-Dependent
Shinn et al. [25] — Reflexion	Agent Systems	System / Architecture	Cloud-Dependent
Liu et al. [21] — RepoBench	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Rozière et al. [15] — Code Llama	Foundation Models	System / Architecture	Local-Capable
Shrivastava et al. [32] — RepoFusion	Agent Systems	System / Architecture	Cloud-Dependent
Lozhkov et al. [18] — StarCoder2	Foundation Models	System / Architecture	Local-Capable

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

Paper	Research Category	Methodology	Deployment Context
Hui et al. [5] — Qwen2.5-Coder	Foundation Models	System / Architecture	Local-Capable
Guo et al. [6] — DeepSeek-Coder	Foundation Models	System / Architecture	Local-Capable
Liu et al. [22] — RepoQA	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Yang et al. [24] — SWE-agent	Agent Systems	System / Architecture	Cloud-Dependent
Sergeyuk et al. [26] — AI coding adoption	Practical Barriers	Empirical Study	Cloud-Dependent
Agarwal et al. [27] — CodeMirage	Practical Barriers	Empirical Study	Cloud-Dependent
Mohsin et al. [28] — LLM security	Practical Barriers	Empirical Study	Deployment-Agnostic
Finkman Noah et al. [30] — CodeCloak	Practical Barriers	System / Architecture	Cloud-Dependent
Chen et al. [23] — evaluation survey	Benchmarks & Evaluation	Survey / Review	Deployment-Agnostic
Chen et al. [33] — CoReQA	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Li et al. [34] — LongCodeU	Benchmarks & Evaluation	Benchmark Paper	Deployment-Agnostic
Liu et al. [35] — CodexGraph	Agent Systems	System / Architecture	Cloud-Dependent
Shen et al. [29] — SecRepoBench	Practical Barriers	Benchmark Paper	Deployment-Agnostic

The analysis reveals several patterns in this classification. Foundation model papers are concentrated in the Cloud-Dependent and Local-Capable categories — by definition, model papers make infrastructure choices. Benchmark papers are almost uniformly classified as Deployment-Agnostic, which reflects a consistent design assumption: benchmarks measure output quality without regard to where or how that output was produced. This assumption is examined critically in the benchmarks section that follows.

C. Findings: RQ1 — Evolution of Code-Oriented LLMs

Ten papers directly address the development of model architectures and training approaches for code-oriented LLMs. The reviewed literature reveals three distinct and sequential phases.

The first phase (2020–2021) centered on representation learning. CodeBERT [11] and GraphCodeBERT [12] established the baseline for understanding code structure through joint pre-training over natural language and programming language. Both were designed for comprehension tasks — code search, documentation, classification — rather than generation, and neither was designed with local deployment as a goal.

The second phase (2021–2023) was defined by generation at scale. Codex [7] demonstrated that fine-tuning on large code corpora produced models capable of generating functionally correct programs from natural language descriptions. CodeGen [13], InCoder [14], and Code Llama [15] each extended this in different directions — open-source availability, infilling, and instruction tuning respectively. CodeT5+ [19] introduced an

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

encoder-decoder architecture with variants ranging from 220M to 16B parameters, with smaller variants capable of running on consumer hardware.

The third phase (2023–2025) is characterized by a shift in design philosophy toward efficiency and accessibility. Qwen2.5-Coder [5], DeepSeek-Coder [6], and StarCoder2 [18] achieve competitive benchmark performance at parameter counts that allow deployment on consumer hardware — from 1B to 34B parameters, with smaller variants runnable on consumer hardware and larger variants accessible through quantization. This reflects a different research goal from the scale-focused second phase: capable code assistance that organizations can deploy privately and locally. All five Local-Capable papers in the review appear in this phase or the transition into it.

Across all three phases, a consistent pattern emerges: each phase addressed limitations exposed by the previous one. Comprehension models could not generate. Generation models performed well on isolated tasks but struggled with repository-level context. The third-phase models narrowed these gaps while also making local deployment feasible. What has not changed across phases is the evaluation context: foundation model papers continue to report performance on benchmarks that do not account for deployment constraints, hardware requirements, or privacy conditions.

The gap this leaves open is concrete. Despite the availability of capable locally deployable models, no reviewed paper evaluates how these models perform under realistic hardware constraints — with limited memory, slower inference, and no access to cloud-side tooling. That boundary is likely task-dependent, and whether a hybrid workflow — local models for execution, cloud models for planning — could bridge it is examined in Section VII-E.

D. Findings: RQ2 — Evaluation Frameworks and Their Limitations

Nine papers directly address the design, scope, and limitations of evaluation frameworks for code-oriented LLMs. The progression of benchmarks reviewed here follows a clear developmental arc, from surface-level similarity metrics toward functional correctness and then toward repository-scale evaluation.

The introduction of HumanEval [7] marked the most significant methodological shift in the period reviewed: replacing surface similarity measures (BLEU, CodeBLEU) with functional correctness assessed by unit tests. This change made evaluation more meaningful, but it also confined the scope of evaluation to isolated, single-function tasks. MBPP [16] and MultiPL-E [17] extended coverage — more Python problems, more programming languages — without addressing this constraint.

RepoBench [21] established a systematic evaluation of repository-level code completion, demonstrating that model performance degrades substantially when tasks require cross-file context rather than single-file completion. RepoQA [22] extended evaluation to long-context code understanding, revealing that models struggle to accurately retrieve and apply information distributed across extended repository structures. SWE-bench [8], derived from real GitHub issues, provides the closest approximation to realistic software engineering evaluation currently available in the literature. Its resolve rates — even for the strongest models — remain low, indicating a substantial gap between benchmark performance on simpler tasks and the ability to handle real engineering problems.

Despite this progression, a structural limitation persists across all reviewed evaluation frameworks. Every major benchmark measures output quality under controlled conditions and says nothing about the circumstances under which that output was produced. Deployment environment, hardware constraints, latency, operational cost, and privacy guarantees are absent from all nine benchmark papers reviewed. Chen et al. [23] identify a related concern: the fragmentation of evaluation metrics means no single framework simultaneously captures correctness, efficiency, readability, and practical usability. The evaluation landscape remains incomplete in ways that matter specifically for deployment decisions.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

The gap is well-defined: no existing benchmark evaluates model performance as a function of deployment context. A researcher or practitioner reading this literature cannot answer the question of whether a locally deployed model is adequate for a given task — not because the models are untested, but because the evaluation frameworks were not designed to ask that question. The next step is the development of deployment-aware benchmarks: evaluation frameworks that incorporate hardware constraints, latency, and operational cost as first-class variables alongside functional correctness. Without them, comparisons between local and cloud-based systems remain informal and unreliable.

E. Findings: RQ3 — Practical Barriers and the Deployment Dimension

Six papers directly address practical barriers to deploying LLMs for software engineering tasks. Four barrier categories emerge from the reviewed literature.

Hallucinations and reliability failures are the most pervasive concern. CodeMirage [27] documents and categorizes hallucination types in LLM-generated code, showing that certain failure modes produce code that is syntactically valid but semantically incorrect — errors that are difficult to detect without comprehensive testing. Empirical evidence from developer adoption studies [4], [26] confirms that such reliability concerns lead developers to verify, double-check, and frequently discard AI-generated outputs, with adoption concentrating around repetitive, lower-stakes tasks while complex or high-consequence code continues to be handled manually.

Security vulnerabilities in generated code represent a compounding risk. Mohsin et al. [28] examine whether code generated by LLMs can be trusted from a security perspective, finding that functional correctness does not guarantee the absence of exploitable vulnerabilities. SecRepoBench [29] extends this concern to repository-level tasks, providing a benchmark specifically designed to evaluate secure code generation in realistic settings. Working code and safe code are not equivalent, and existing benchmarks that measure functional correctness alone do not capture this distinction.

Trust gaps are quantified by the empirical studies of Sergeyuk et al. [26] and Peng et al. [4]. Both document developer behavior in practice: verification is common, trust in AI-generated outputs is selective, and the productivity gains documented under controlled conditions are narrower in real development workflows. The empirical picture that emerges is one of selective rather than broad adoption, with a persistent gap between benchmark performance and trust in production use.

Privacy risks are architecturally distinct from the preceding three barriers. Hallucinations, security issues, and trust gaps are technical problems that can be mitigated through better models, review practices, or static analysis tools. The risk of exposing proprietary code when using cloud-based coding assistants is structural — it is a consequence of the cloud deployment model itself. Finkman Noah et al. [30] address this risk directly with CodeCloak, a system designed to protect code confidentiality by obfuscating code before it is transmitted to a cloud-based model. The need for such a tool illustrates the core tension: cloud deployment provides access to the most capable models, but it comes with inherent confidentiality risks that cannot be resolved within the cloud model and which are a significant barrier in regulated or proprietary development environments.

The deployment context distribution documented in Section VII-A — only 17% of reviewed papers explicitly supporting or evaluating local deployment — is directly relevant here. The privacy barrier is the one most resistant to technical mitigation within cloud infrastructure, yet it is also the barrier that local deployment most directly resolves. No reviewed benchmark framework incorporates privacy or deployment conditions as evaluation dimensions, leaving a gap between the research literature and the practical decision that organizations face when choosing between cloud and local AI coding assistance.

Taken together, the four barrier categories point toward a gap that is organizational as much as technical. The research community has studied what these tools can do under ideal conditions. It has studied far less what

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

happens when those conditions do not hold — when the hardware is constrained, when the codebase cannot leave the local environment, when the output must be trusted without extensive verification.

For a reader seeking to act on this literature, three directions stand out. First, organizations evaluating AI coding tools should treat privacy requirements as a primary filter before assessing performance — the deployment model must be acceptable before benchmark scores become relevant. Second, researchers should prioritize empirical studies of developer behavior under local deployment conditions, since the existing adoption literature is almost entirely cloud-focused. Third, a hybrid workflow — local models handling implementation-level tasks, cloud models reserved for high-level reasoning — represents a practical and currently testable path between privacy requirements and capability needs. None of these directions has been addressed systematically in the reviewed literature.

VIII. DISCUSSION

The literature reviewed in this paper traces a clear progression — from models that understood code to models that generate it, from isolated task evaluation to repository-level benchmarks, and from standalone generation to agent-based systems that interact with tools and environments. That progression is real and the advances are genuine. But looking across the full body of work, a consistent pattern emerges that individual papers do not always make explicit.

Scale and data have driven most of the measurable progress. Models got better largely because they got bigger and were trained on more code. Benchmark scores improved accordingly. What did not improve at the same rate was the ability to reason — to navigate unfamiliar codebases, track dependencies across files, and produce reliable output under conditions the model has not seen before. The gap between HumanEval performance and SWE-bench performance is not a benchmarking artifact. It reflects something real about how these models solve problems, and it has persisted across multiple generations of increasingly capable systems.

Benchmarks have shaped the field in ways worth being honest about. Each new evaluation framework revealed limitations the previous one had obscured. HumanEval exposed the inadequacy of surface similarity metrics. SWE-bench exposed the inadequacy of isolated task evaluation. This is progress, but it also means reported improvements need to be read carefully — a model that scores well on the current benchmark may simply be good at the current benchmark. The field has not yet settled on an evaluation framework that fully captures what capable software engineering assistance actually looks like in practice.

The limitations discussed in Section VI — hallucinations, security vulnerabilities, trust gaps, and privacy risks — are not separate problems. They share a common root. Models that rely heavily on pattern recognition over genuine reasoning will produce outputs that look correct more often than they are correct. Hallucinations, security gaps, and trust issues are all symptoms of that same underlying weakness. Scaling has not resolved this, and there is limited evidence that it will on its own.

What the literature has been slower to examine is a different set of questions — ones that concern not what models can do but under what conditions they can be used. Most research reviewed here assumes cloud-based infrastructure. Model comparisons, benchmark evaluations, and agent-based systems are all designed and tested in that context. The hardware constraints, latency profiles, cost structures, and privacy implications of local deployment are largely absent from how progress is measured and reported. This is not a minor methodological gap. It determines whether the tools being developed are accessible to the full range of developers and organizations that might benefit from them.

Locally deployed models have received comparatively little systematic attention despite the fact that capable open-source models now exist at parameter counts that run on consumer hardware. As the overview in Section VII documents, only 17% of reviewed papers explicitly support or evaluate local deployment — a distribution that reflects how consistently the field has treated cloud infrastructure as the default context. Whether these models are sufficient for practical use — and specifically whether they handle certain categories of tasks well

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

enough to reduce cloud dependency — remains an open empirical question. This question is particularly relevant given the three-phase progression documented in Section III-D, where the most recent phase is defined precisely by a shift toward efficiency and local accessibility. The planning versus execution distinction is one angle worth examining here. Tasks that require high-level reasoning and architectural judgment may place fundamentally different demands on a model than tasks focused on concrete implementation. If local models hold up adequately on execution-oriented work, a hybrid workflow becomes viable — and that possibility has practical consequences for cost, privacy, and accessibility that the current literature does not address.

The field has made significant progress on what it has chosen to measure. The more important question now is whether it is measuring the right things.

IX. Challenges and Future Research Priorities

The review carried out in the preceding sections points toward several concrete directions where the field needs to move. These are not speculative suggestions — each one follows directly from a gap identified in the literature reviewed here.

A. Develop Deployment-Aware Evaluation Frameworks

As established in Section IV, no current benchmark incorporates deployment environment, hardware constraints, latency, or operational cost as evaluation variables. Two systems producing identical output quality may differ substantially in cost, speed, and the privacy guarantees they offer — yet existing benchmarks treat them as equivalent. Developing evaluation frameworks that incorporate these dimensions would make comparisons between local and cloud-based systems meaningful in a way that current evaluations simply are not.

B. Conduct Systematic Local vs Cloud Comparative Studies

Most existing work evaluates models in isolation or compares architectures under identical infrastructure assumptions. Direct comparison across deployment types — using the same tasks, the same repositories, and the same evaluation criteria — would produce findings immediately relevant to developers and organizations making real deployment decisions. Such comparisons should go beyond aggregate performance and examine where the gaps actually lie across task categories, since a uniform performance deficit tells a different story than one concentrated in specific task types.

C. Investigate the Planning vs Execution Task Distinction

Software engineering tasks are not uniform — some require high-level reasoning and architectural judgment, others involve concrete implementation given a clear specification. Whether local and cloud models differ in competency across these categories in ways that aggregate benchmarks obscure is an open empirical question. Testing this requires controlled experiments that vary both task type and deployment context simultaneously: the same model, the same repository, measured on planning-heavy and execution-heavy tasks under both local and cloud conditions. That experimental design does not exist in the reviewed literature and would produce findings directly relevant to deployment decisions.

D. Examine Repository Reasoning Under Hardware Constraints

Retrieval-augmented approaches and graph-based representations have shown promise for repository-level reasoning, but neither has been evaluated under local deployment constraints. The approaches reviewed in Sections IV and V were built and tested under infrastructure assumptions that do not reflect the constraints developers actually face. Whether these approaches remain viable when computational resources are limited — when inference is slower, memory is constrained, and graph construction overhead matters — is an open

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

question. It is not enough to show that an approach works on powerful infrastructure. It needs to work on hardware that developers actually have access to.

E. Treat Privacy as a First-Class Research Concern

Privacy risks from cloud-based tools are documented in the literature but remain peripheral to how systems are designed and evaluated. As locally deployed models improve, understanding the practical tradeoffs between local and cloud deployment — in terms of performance, cost, and privacy guarantees — will become increasingly relevant. This is particularly true for enterprise and regulated environments where cloud-based tools may not be viable regardless of their benchmark performance. Privacy warrants treatment as a primary research variable rather than a deployment consideration addressed after the fact.

X. Limitations of This Review

No review is exhaustive, and this one is no exception. Several boundaries were set deliberately, and others reflect the constraints of the literature search and the scope of the review's central questions.

The search was conducted as a structured rather than exhaustive review, and some relevant work may have been missed, particularly recent preprints that had not yet accumulated citations at the time of writing. Paper selection was guided by relevance to the three organizing questions rather than by a predefined protocol.

The review focuses exclusively on English-language models and English-language code contexts. Several areas are deliberately out of scope. Non-English programming contexts and multilingual codebases are not covered. Neither are LLM testing methodologies, formal verification approaches, or the training and infrastructure considerations behind building these models. Each of these is a legitimate research area.

Framing the review around local versus cloud deployment meant other deployment contexts received less attention. Edge computing, federated learning, and mobile inference are all relevant to the broader question of deployment constraints — they simply fall outside the boundary this review drew. That boundary was deliberate, but researchers working in those areas should read the coverage accordingly.

XI. Conclusion

This review has examined the current state of large language models applied to software engineering, tracing their development from early representation-focused models through to modern generation and reasoning systems. Along the way it has looked at how evaluation frameworks have evolved, where retrieval and agent-based approaches have made genuine contributions, and what limitations persist across the field despite significant progress in model capability.

The central finding is straightforward. The research community has gotten very good at improving benchmark scores. It has been less successful at closing the gap between those scores and what capable software engineering assistance actually requires in practice. Repository-level reasoning remains difficult. Hallucinations and security vulnerabilities persist. Trust gaps have not closed. The evaluation frameworks used to measure progress have consistently lagged behind the complexity of real development environments.

What the literature has largely not addressed is the practical dimension of deployment. Most research assumes cloud-based infrastructure and optimizes accordingly. Privacy risks, hardware constraints, operational costs, and the tradeoffs between local and cloud-based systems have received limited systematic attention despite being consequential for a large portion of real development contexts. Locally deployed open-source models have reached a level of capability that makes them worth taking seriously, but rigorous comparative evaluation under realistic conditions remains scarce.

That is the gap this review points toward. Whether locally deployed models are sufficient for practical software engineering tasks — and specifically whether they handle execution-oriented work well enough to support a hybrid workflow alongside cloud-based tools — is an empirical question the current literature does not

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

answer. Addressing it will require new evaluation frameworks, controlled comparisons across deployment types, and a willingness to measure what actually matters to developers rather than what is easiest to benchmark. The tools exist. The research agenda is clear. Acting on it is the work that remains.

REFERENCES

[1]. GitHub, "GitHub Copilot." [Online]. Available: https://github.com/features/copilot (accessed Oct. 25, 2026).
[2]. Anthropic, "Claude Code." [Online]. Available: https://www.anthropic.com/claude-code (accessed Oct. 25, 2026).
[3]. Google, "Gemini CLI," GitHub repository. [Online]. Available: https://github.com/google-gemini/gemini-cli (accessed Oct. 25, 2026).
[4]. S. Peng et al., "The impact of AI on developer productivity: Evidence from GitHub Copilot," *arXiv preprint* arXiv:2302.06590, 2023. [Online]. Available: https://arxiv.org/abs/2302.06590
[5]. B. Hui et al., "Qwen2.5-Coder technical report," *arXiv preprint* arXiv:2409.12186, 2024. [Online]. Available: https://arxiv.org/abs/2409.12186
[6]. D. Guo et al., "DeepSeek-Coder: When the large language model meets programming—The rise of code intelligence," *arXiv preprint* arXiv:2401.14196, 2024. [Online]. Available: https://arxiv.org/abs/2401.14196
[7]. M. Chen et al., "Evaluating large language models trained on code," *arXiv preprint* arXiv:2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374
[8]. C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.
[9]. B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ., Keele, U.K., Tech. Rep. TR/SE-0401, 2004.
[10]. J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quarterly*, vol. 26, no. 2, pp. xiii–xxiii, 2002.
[11]. Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in *Findings Assoc. Comput. Linguistics: EMNLP*, 2020.
[12]. D. Guo et al., "GraphCodeBERT: Pre-training code representations with data flow," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.
[13]. E. Nijkamp et al., "CodeGen: An open large language model for code with multi-turn program synthesis," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.
[14]. D. Fried et al., "InCoder: A generative model for code infilling and synthesis," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.
[15]. B. Rozière et al., "Code Llama: Open foundation models for code," *arXiv preprint* arXiv:2308.12950, 2023. [Online]. Available: https://arxiv.org/abs/2308.12950
[16]. J. Austin et al., "Program synthesis with large language models," *arXiv preprint* arXiv:2108.07732, 2021. [Online]. Available: https://arxiv.org/abs/2108.07732
[17]. F. Cassano et al., "MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation," *IEEE Trans. Softw. Eng.*, vol. 49, no. 7, pp. 3675–3691, 2023.
[18]. A. Lozhkov et al., "StarCoder 2 and The Stack v2: The next generation," *arXiv preprint* arXiv:2402.19173, 2024. [Online]. Available: https://arxiv.org/abs/2402.19173
[19]. Y. Wang et al., "CodeT5+: Open code large language models for code understanding and generation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2023.
[20]. S. Lu et al., "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," in *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks*, 2021.
[21]. T. Liu, C. Xu, and J. McAuley, "RepoBench: Benchmarking repository-level code auto-completion systems," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden

International Journal of Computer Science and Mobile Computing
A Monthly Journal of Computer Science and Information Technology

Sead Mašetić *et al*, Volume 15, Issue 11, November 2026, pg. 1-22
(An Open Accessible, ISI Indexed, Fully Refereed and Peer Reviewed Journal)
ISSN: 2320-088X
Impact Factor: 7.056

[22]. J. Liu et al., "RepoQA: Evaluating long context code understanding," *arXiv preprint* arXiv:2406.06025, 2024. [Online]. Available: https://arxiv.org/abs/2406.06025
[23]. L. Chen et al., "A survey on evaluating large language models in code generation tasks," *arXiv preprint* arXiv:2408.16498, 2024. [Online]. Available: https://arxiv.org/abs/2408.16498
[24]. J. Yang et al., "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 37, 2024.
[25]. N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 36, 2023.
[26]. A. Sergeyuk et al., "Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward," *Inf. Softw. Technol.*, vol. 178, Art. no. 107610, 2025.
[27]. V. Agarwal et al., "CodeMirage: Hallucinations in code generated by large language models," *arXiv preprint* arXiv:2408.08333, 2024. [Online]. Available: https://arxiv.org/abs/2408.08333
[28]. A. Mohsin et al., "Can we trust large language models generated code? A framework for in-context learning, security patterns, and code evaluations across diverse LLMs," *arXiv preprint* arXiv:2406.12513, 2024. [Online]. Available: https://arxiv.org/abs/2406.12513
[29]. C. Shen et al., "SecRepoBench: Benchmarking code agents for secure code completion in real-world repositories," *arXiv preprint* arXiv:2504.21205, 2025. [Online]. Available: https://arxiv.org/abs/2504.21205
[30]. A. Finkman Noah et al., "CodeCloak: A method for evaluating and mitigating code leakage by LLM code assistants," *arXiv preprint* arXiv:2404.09066, 2024. [Online]. Available: https://arxiv.org/abs/2404.09066
[31]. F. Zhang et al., "RepoCoder: Repository-level code completion through iterative retrieval and generation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2023.
[32]. D. Shrivastava et al., "RepoFusion: Training code models to understand your repository," *arXiv preprint* arXiv:2306.10998, 2023. [Online]. Available: https://arxiv.org/abs/2306.10998
[33]. J. Chen et al., "CoReQA: Uncovering potentials of language models in code repository question answering," *arXiv preprint* arXiv:2501.03447, 2025. [Online]. Available: https://arxiv.org/abs/2501.03447
[34]. J. Li et al., "LongCodeU: Benchmarking long-context language models on long code understanding," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2025.
[35]. X. Liu et al., "CodexGraph: Bridging large language models and code repositories via code graph databases," in *Proc. Conf. Nations Americas Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL)*, 2025.

© 2026, IJCSMC All Rights Reserved, ZAIN Publications, Fridhemsgatan 62, 112 46, Stockholm, Sweden