LLM-Based Agents for SE: Key Ideas & Open Questions

Based on survey by Junwei Li et al. TOSEM'25
Research Roadmap by Ahmed Hassan et al. TOSEM

Danny Dig

University of Colorado Boulder

What do you most want to get out of today’s discussion?

Today’s goals

Reconstruct the **conceptual map** of LLM-based agents in SE

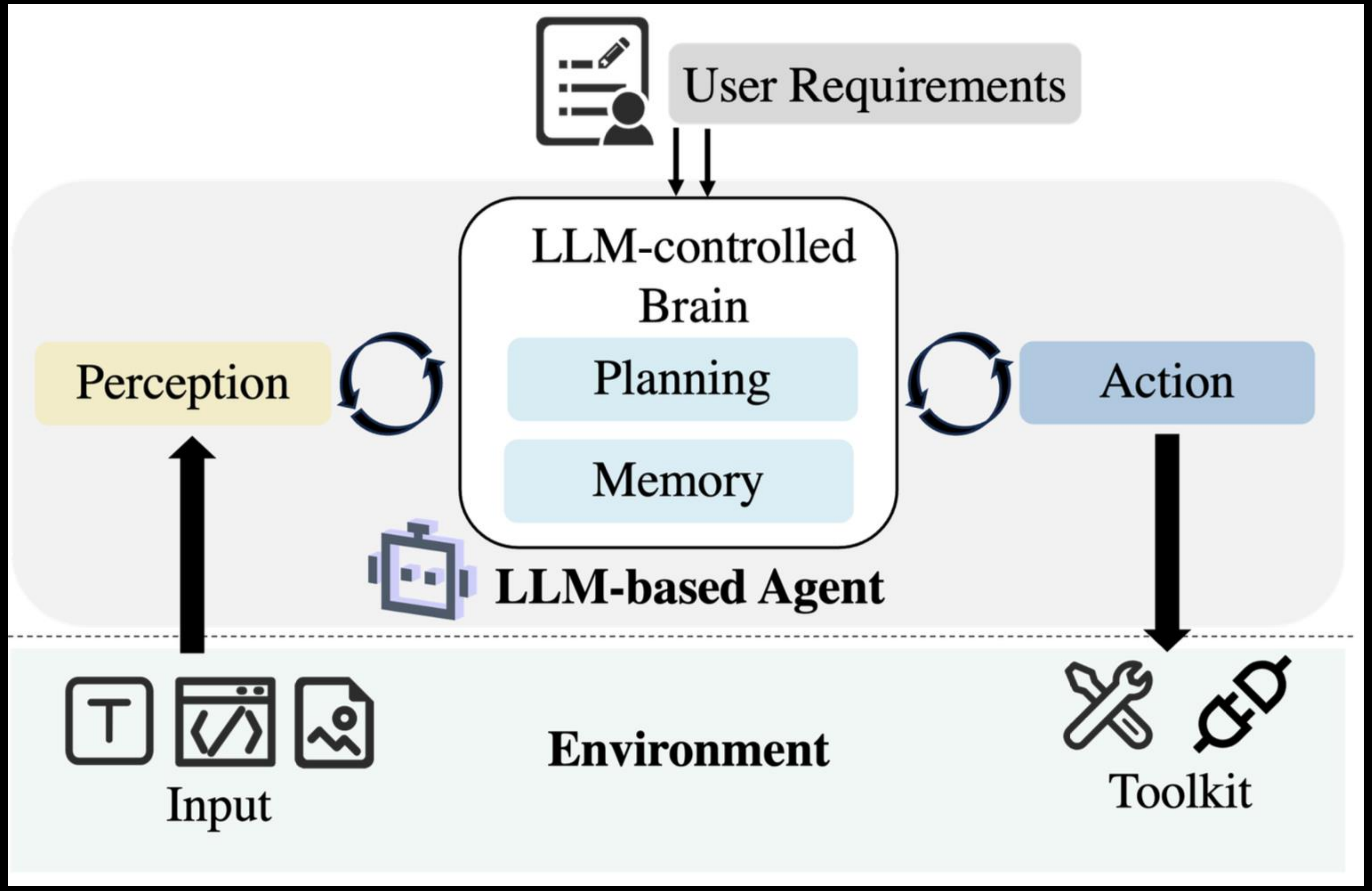
Identify **where agentic systems already work well vs. poorly**

Spot **gaps, blind spots, and weak assumptions** in current research

Translate **research opportunities → concrete project ideas**

What Do the Authors Mean by “Agent”?

Beyond single-prompt LLM usage



What distinguishes an “agent” from a clever prompt or script?

Where is the boundary fuzzy in this paper?

Agent Taxonomy (Architectural View)

Single-agent vs multi-agent

Centralized vs decentralized control

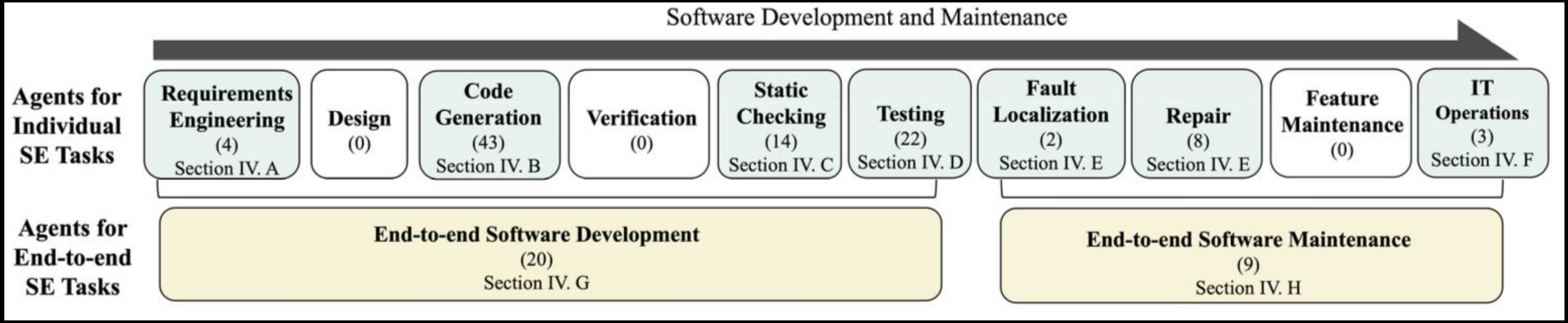
Role-based agents

Tool-augmented agents

Which architectures feel overrepresented in the literature? Why?

Which architectures seem underexplored or impractical today?

Software Development Lifecycle Coverage Map


Software Development and Maintenance
Agents for Individual SE Tasks
Requirements Engineering (4) Section IV. A
Design (0)
Code Generation (43) Section IV. B
Verification (0)
Static Checking (14) Section IV. C
Testing (22) Section IV. D
Fault Localization (2) Section IV. E
Repair (8) Section IV. E
Feature Maintenance (0)
IT Operations (3) Section IV. F
Agents for End-to-end SE Tasks
End-to-end Software Development (20) Section IV. G
End-to-end Software Maintenance (9) Section IV. H

Where agents are commonly applied
Where coverage is thin (?)

Which SDLC phases dominate agent research?

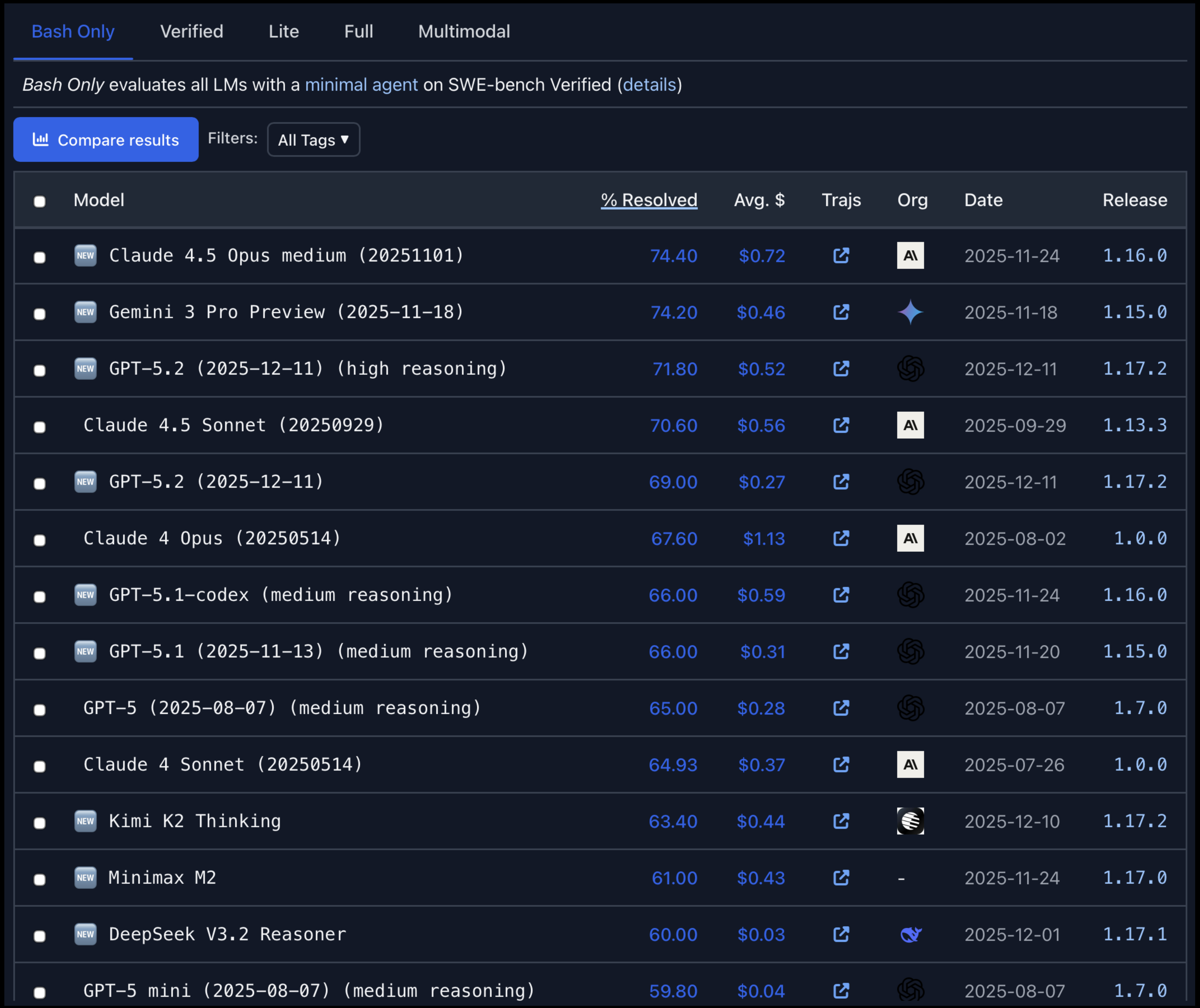
Is this because of technical feasibility, data availability, or research convenience?

Deep Dive: Strong Coverage Areas

Code generation / repair
Test generation
Bug fixing
Program comprehension

What assumptions about software development are baked into these successes?
Do these agents help individuals, teams, or organizations?

SWE-Bench



Bash Only | Verified | Lite | Full | Multimodal

Bash Only evaluates all LMs with a minimal agent on SWE-bench Verified (details)

Compare results Filters: All Tags ▾

	Model	% Resolved	Avg. $	Trajs	Org	Date	Release
☐	NEW Claude 4.5 Opus medium (20251101)	74.40	$0.72	↗	A\	2025-11-24	1.16.0
☐	NEW Gemini 3 Pro Preview (2025-11-18)	74.20	$0.46	↗	✦	2025-11-18	1.15.0
☐	NEW GPT-5.2 (2025-12-11) (high reasoning)	71.80	$0.52	↗		2025-12-11	1.17.2
☐	Claude 4.5 Sonnet (20250929)	70.60	$0.56	↗	A\	2025-09-29	1.13.3
☐	NEW GPT-5.2 (2025-12-11)	69.00	$0.27	↗		2025-12-11	1.17.2
☐	Claude 4 Opus (20250514)	67.60	$1.13	↗	A\	2025-08-02	1.0.0
☐	NEW GPT-5.1-codex (medium reasoning)	66.00	$0.59	↗		2025-11-24	1.16.0
☐	NEW GPT-5.1 (2025-11-13) (medium reasoning)	66.00	$0.31	↗		2025-11-20	1.15.0
☐	GPT-5 (2025-08-07) (medium reasoning)	65.00	$0.28	↗		2025-08-07	1.7.0
☐	Claude 4 Sonnet (20250514)	64.93	$0.37	↗	A\	2025-07-26	1.0.0
☐	NEW Kimi K2 Thinking	63.40	$0.44	↗		2025-12-10	1.17.2
☐	NEW Minimax M2	61.00	$0.43	↗	-	2025-11-24	1.17.0
☐	NEW DeepSeek V3.2 Reasoner	60.00	$0.03	↗		2025-12-01	1.17.1
☐	GPT-5 mini (2025-08-07) (medium reasoning)	59.80	$0.04	↗		2025-08-07	1.7.0

Weak or Missing Coverage Areas

Highlighted gaps:
Requirements engineering
Architectural decision-making
Socio-technical coordination
DevOps incident response
Long-term maintenance & technical debt

Pick one SDLC phase here. Why is it hard for agents today?

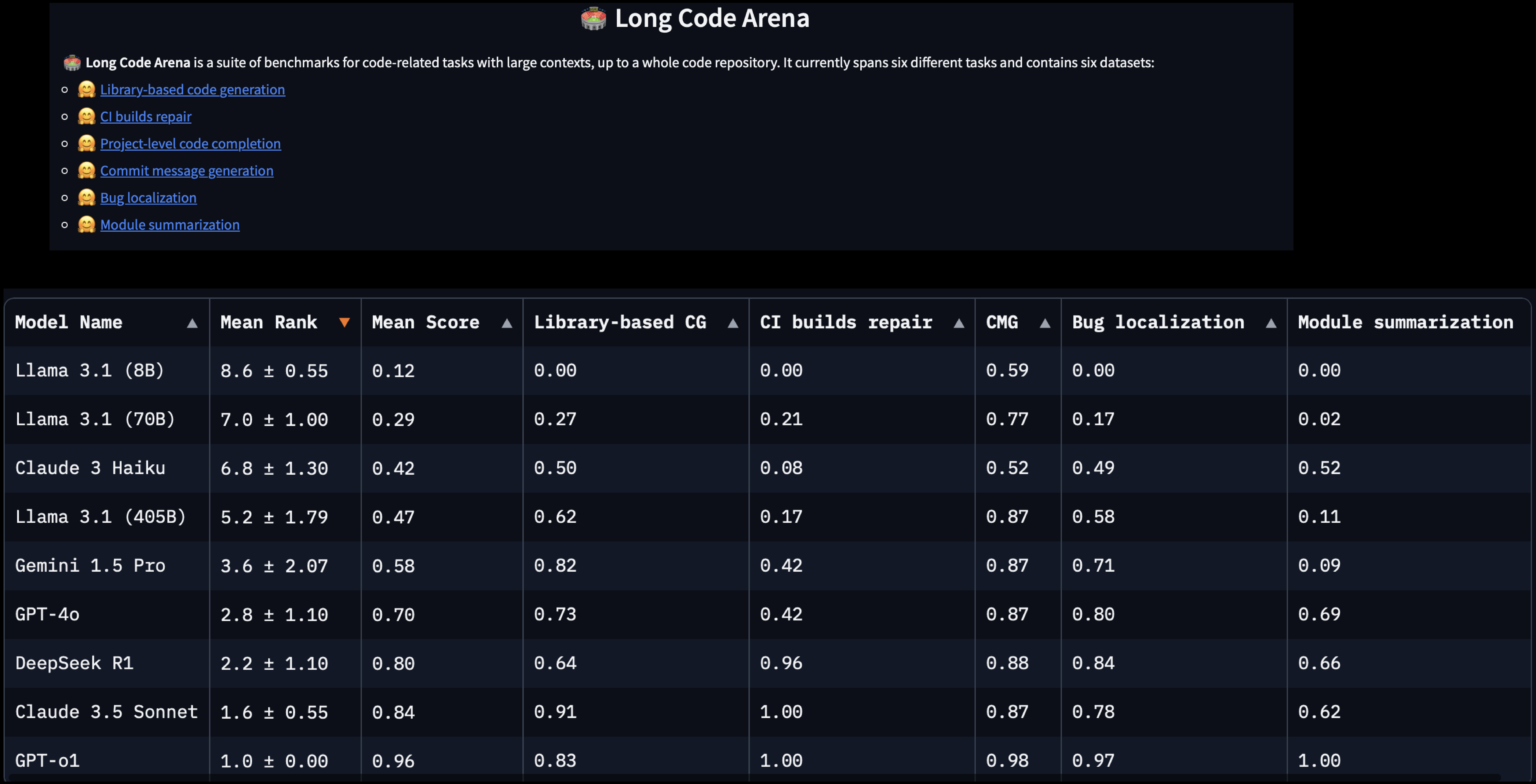
Is the blocker technical, organizational, or conceptual?

Which of these missing coverage areas trigger your imagination for building an agent for?

Evaluation: How Are Agents Being Measured?

Benchmarks
Task success rates
Code-centric metrics
Synthetic tasks

What is missing from current evaluation approaches?
How would you evaluate an agent working on a 3-month project?



🏟️ Long Code Arena

🏟️ **Long Code Arena** is a suite of benchmarks for code-related tasks with large contexts, up to a whole code repository. It currently spans six different tasks and contains six datasets:

- 🤗 [Library-based code generation]
- 🤗 [CI builds repair]
- 🤗 [Project-level code completion]
- 🤗 [Commit message generation]
- 🤗 [Bug localization]
- 🤗 [Module summarization]

Model Name	Mean Rank	Mean Score	Library-based CG	CI builds repair	CMG	Bug localization	Module summarization
Llama 3.1 (8B)	8.6 ± 0.55	0.12	0.00	0.00	0.59	0.00	0.00
Llama 3.1 (70B)	7.0 ± 1.00	0.29	0.27	0.21	0.77	0.17	0.02
Claude 3 Haiku	6.8 ± 1.30	0.42	0.50	0.08	0.52	0.49	0.52
Llama 3.1 (405B)	5.2 ± 1.79	0.47	0.62	0.17	0.87	0.58	0.11
Gemini 1.5 Pro	3.6 ± 2.07	0.58	0.82	0.42	0.87	0.71	0.09
GPT-4o	2.8 ± 1.10	0.70	0.73	0.42	0.87	0.80	0.69
DeepSeek R1	2.2 ± 1.10	0.80	0.64	0.96	0.88	0.84	0.66
Claude 3.5 Sonnet	1.6 ± 0.55	0.84	0.91	1.00	0.87	0.78	0.62
GPT-o1	1.0 ± 0.00	0.96	0.83	1.00	0.98	0.97	1.00

Human–Agent Interaction Assumptions

Many systems assume:

- Clear task boundaries
- Well-specified goals
- Minimal human negotiation

How realistic are these assumptions in real software teams?
What human behaviors are ignored?

Section 6: Research Opportunities

Scalability & robustness

Trust, safety, alignment

Long-horizon reasoning

Human-in-the-loop agents

Real-world deployment

Research Opportunities → Project Seeds

Which opportunity excites you most—and why?

What would a small but meaningful prototype look like?

What would you NOT try to solve in a semester?

High-Impact Project-Trigger Questions

- **Humans in the Loop**

Should agents adapt to individual developer styles?
How do we prevent “over-automation”?

- **Evaluation**

What would a **qualitative** benchmark for agents look like?
How do we measure trust or usefulness?

Evaluation & Benchmarks: What Does "Success" Mean for Agents?

- Is it possible to create a benchmark that evaluates **both intermediate reasoning states and final outcomes**?
- How can we **standardize benchmarks** so agents with different architectures are comparable?
- How can we rigorously evaluate agents on **long-running software projects**, not isolated tasks?
- What is the right metric for success on real projects?
- Code quality?
- Deployment frequency?
- Bug reduction?
- Developer productivity?
- How can evaluation frameworks capture **long-term qualities** like maintainability, architectural soundness, or developer trust?
- How do we evaluate **autonomy** beyond task completion?

Architecture Tradeoffs: When Is “Agentic” Actually Worth It?

- Why do **simpler ML or prompt-based approaches** sometimes outperform more sophisticated agentic ones?
- What performance and **cost tradeoffs** exist between:
- Multi-agent systems
- Single agents with tool access
- Under what conditions is **multi-agent worth the overhead**?
- Which software activities truly benefit from **job separation across agents**?
- To what extent do multi-agent systems outperform strong baselines *once coordination overhead is included*?
- What are the tradeoffs between different **memory designs** (vectorized vs natural language)?

Memory, Context, and Long-Term Stability

- How stable are agent behaviors as **memories grow over long periods**?
- How should an agent decide when it has **enough context to act**, versus when more context adds noise?
- How is information being measured in these systems?
- Can ideas from **information theory** help with efficient context distillation?
- What are the tradeoffs between:
- Storing information efficiently
- Conveying it clearly for reasoning
- How do we prevent **context window saturation** from degrading performance?

Autonomy, Human Control, and Responsibility

- What responsibilities should **remain human-driven** in an agent-assisted SE pipeline?
- How do we prevent **compounding errors**, especially in agents with reflection or self-improvement?
- What can agents do to prevent the **reinforcement of wrong assumptions**?
- If an agent introduces a vulnerability:
- Who is responsible?
- How does the system justify its actions?
- What is the right balance between **agent autonomy and human oversight** in long-running projects?
- Could agents adapt to—or reshape—**company-specific coding practices**, and who decides?

Emergence, Roles, and the Future of Software Engineers

- To what extent are agent capabilities truly **emergent**, versus prompt-engineered?
- How are agent roles created: Prompting? Rules? Fine-tuning? Tool feedback?
- How long will LLM capabilities continue improving rapidly—and what happens when they plateau?
- Will agentic systems **fully replace software engineers**, or are we seeing diminishing returns?
- If base LLM improvements still dominate gains, what does that imply for agent research?

Agentic Software Engineering: Foundational Pillars and a Research Roadmap

Ahmed Hassan et al.

From Agentic Coding to Agentic Software Engineering (SE 3.0)

Memory refresh

- SE 1.0: Manual coding
- SE 2.0: AI-augmented coding (Copilot, autocomplete)
- **SE 3.0: Goal-agentic systems** executing multi-step SE tasks
- Core tension: **speed vs trust**

Why do the authors argue that today’s tools are still “agentic coding,” not SE?
What *breaks* when we scale from one agent → many agents?

Agency vs Autonomy: A Critical Distinction

Memory refresh

- **Agency**: execute a given plan
- **Autonomy**: formulate the plan itself
- Analogy to SAE self-driving levels (Levels 0–5)
- Paper focuses on **Level 3 (Goal-Agentic SE)**

Why is Level-3 the “danger zone” for software engineering?
What risks appear *before* full autonomy (Levels 4–5)?

The Central Claim: Duality of SE

Two simultaneous modalities:

- **SE for Humans (SE4H)** → humans as *Agent Coaches*
- **SE for Agents (SE4A)** → predictable, structured agent environments

Same four pillars, different manifestations:

- Actors, Processes, Tools, Artifacts

What existing SE assumptions become invalid when agents are first-class actors?
Is this duality descriptive, or a prescription the authors are arguing *for*?

Why Today’s IDEs and Workflows Fail

IDEs optimized for human cognition

Chat-based prompting is:

- Ephemeral
- Non-auditable
- Non-reproducible
- Leads to review bottlenecks and “merge-unready” code

Which of your own workflows already show these cracks?
Could better models alone fix this—or is structure unavoidable?

Structured Agentic Software Engineering (SASE)

SASE = structured human–agent collaboration

Core idea: **artifacts as the interface**

Replace Informal prompts → version-controlled artifacts

- Goal: make agentic SE **auditable, reproducible, and scalable**

What feels most radical: the artifacts, or the human role shift?

Two New Workbenches: ACE & AEE

ACE (Agent Command Environment)

- Human-optimized
- Orchestration, review, mentorship

AEE (Agent Execution Environment)

- Agent-optimized
- Parallelism, raw tools, monitoring

What would *you* want to see in an ACE that IDEs don’t provide?
Why is it dangerous to optimize tools for humans *and* agents at once?

New Artifacts: Making Trust Explicit

Key SASE artifacts:

- **BriefingScript** – intent + success criteria
- **LoopScript** – workflow & rigor
- **MentorScript** – norms & best practices
- **Consultation Request Pack (CRP)** – agent → human escalation
- **Merge-Readiness Pack (MRP)** – evidence-based deliverable

Which artifact feels most immediately useful?
Which feels hardest to design well—and why?

Engineering Activities, Not Just Tools

Six core activities:

1. Briefing Engineering (BriefingEng)
2. Agentic Loop Engineering (ALE)
3. AI Teammate Mentorship Engineering (ATME)
4. Agentic Guidance Engineering (AGE)
5. AI Teammate Lifecycle Engineering (ATLE)
6. AI Teammate Infrastructure Engineering (ATIE)

Which of these is closest to an existing SE discipline you know?
Which represents genuinely *new* research territory?

The Bitter Lesson (Revisited)

Sutton’s Bitter Lesson: scale > human-encoded knowledge

Authors’ position:
- Structure ≠ fighting scale
- Structure enables **reliable scaling in complex domains**

Humans provide: Strategy, Ethics, Guardrails, Judgment

Is SASE aligned with, or in tension with, the Bitter Lesson?
When should humans impose structure—and when should they step back?

Research & Education Implications (Project Bridge)

Research shifts:

- From code → coordination, observability, memory, trust

Education shifts:

- From “writing code” → **mentoring and orchestrating agents**

The human becomes:

- Conductor, not typist

Which SASE component would make the *best class project*?
If we trained students this way, what skills would we stop teaching?

Transparency, Observability, and Trust in Agent Decisions

- What level of observability is necessary for humans to **trust, audit, and intervene** in agent decisions without causing cognitive overload?
- How should we trace and explain agent actions across **multiple iterations, long-running workflows, and N-to-N agent collaborations**?
- Does increased structure (e.g., scripts, logs, artifacts) genuinely improve trust, or does it merely create the *appearance* of rigor?
- How much transparency is “enough” before it starts degrading agent performance or human usability?

Context, Memory, and Long-Term Alignment

- When does long-term memory meaningfully improve agent performance, and when does it introduce noise or misalignment?
- How should agents balance **short-term efficiency vs. long-term alignment**, and does this tradeoff vary by task or domain?
- Are Briefing Packs better viewed as **persistent memory** or as **retrieval targets** activated selectively?
- How should retrieval-based mechanisms (e.g., RAG) be designed, evaluated, and governed in agentic workflows?

Human–Agent Responsibility, Oversight, and Accountability

- Who is legally and ethically responsible when autonomous agents introduce faults or regressions?
- How do we prevent **automation complacency** in human reviewers as agents become more capable?
- Can agents reliably decide when to escalate issues to humans—and how do we validate that behavior?
- Who authorizes and validates control artifacts like MentorScripts and LoopScripts?

Reproducibility, Versioning, and Engineering Rigor

- How can we guarantee reproducibility when LLM outputs are inherently stochastic?
- What is the right **granularity of version control** for agent-generated artifacts, reasoning traces, and decisions?
- How do we benchmark “merge readiness” or correctness in a standardized, repeatable way?
- How do we ensure agents don’t reinforce bad or outdated engineering norms over time?

Cost, Scalability, and Practical Adoption

- Can small teams, startups, and open-source projects realistically afford agentic infrastructure?
- What is the cost overhead of maintaining SASE artifacts relative to existing workflows?
- Will agentic systems ever see widespread adoption given inference and orchestration costs?
- How does SASE affect real-world metrics like commit rates, review latency, and regression frequency?

Human Roles, Skills, and Team Dynamics

- How does the role of developers change when humans act primarily as specifiers rather than implementers?
- How do we preserve creativity and solution diversity when agents optimize toward “optimal” outputs?
- How should senior engineers be trained when agents take on entry-level tasks?
- Will agentic SE change who enters the field, and could it broaden participation?