Blitzy System 2 AI Platform: Topping SWE-bench Verified

Domain Specific Context Engineering paired with Extended Inference Time Validation overcomes existing barriers in LLM driven Software Development

Neeraj Deshmukh, Siddhant Pardeshi, Brian Elliott, Jack Blundin, Advika Sadineni, Yash Bolishetti, David Rome, Simon Mead, and Advait Sadineni

Abstract

Large language models (LLMs) for software engineering have advanced rapidly, yet current coding copilots remain constrained by finite context and limited autonomous validation. AI-powered IDE systems typically process a slice of a repository, leaving them unable to anticipate the downstream ramifications of changes across millions of lines of code. Simultaneously, the perceived need for 'in-session code responses' has limited the progress of extended inference time compute validation techniques.

We introduce Blitzy – a system-2 agentic platform designed for enterprise scale and enterprise grade code generation. Blitzy makes 2 key trade-offs. First, Blitzy trades inference latency (time for the end user) for expandable usable context to operate effectively on large scale codebases (tens of millions of lines of code). Second, Blitzy trades inference-time costs for higher quality, pre-validated code.

The Blitzy platform can ingest entire codebases (multiple repositories) to extract a unified technical specification, and implement autonomously tested code changes aligned with the global understanding.

Blitzy achieves an unprecedented Pass@1 success rate of 86.8% on the SWE-bench Verified benchmark, exceeding the prior best by 13%. Even more remarkably, we accomplish this using Blitzy's standard deployed platform without any scaffolding, hints, best-of-k attempts, or other task-specific refinements; thus providing reproducibility for any Blitzy subscriber.

Our success demonstrates the power of these trade-offs to enable meaningful progress on real-world software engineering tasks, while also highlighting the need for future benchmarks that better capture repository-scale reasoning and long-term codebase health.

1. Introduction to SWE-bench Verified

SWE-bench Verified, developed by Princeton researchers and released by OpenAI in August 2024, is a human-curated subset of 500 tasks selected from the 2294 instances in the original SWE-bench benchmark. It filters out issues that were deemed ambiguous, underspecified, or effectively unsolvable; and improves evaluation accuracy and reliability to provide a more stable, reliable and yet realistic evaluation set. The included problems each provide a short natural-language description taken from a real GitHub bug report, yet require models to produce a correct patch while reasoning over the full repository context, requiring pre- and post-patch tests to receive a passing grade.

1.1 Significance of SWE-bench Verified Benchmarks

SWE-bench Verified provides a clearer and consistent signal in measuring reasoning capability, avoids many of the pitfalls associated with noisy or unsolvable tasks, and aligns with the benchmark most widely used by leading labs in reporting state-of-the-art results. Consequently, it has quickly established itself as the gold-standard benchmark for autonomous code generation.

High scores on this benchmark are influential: they are widely cited in academic papers, drive product positioning across industry, and shape perceptions of progress in autonomous software development (e.g., Devin parlayed a 13.96% score on SWE-bench Verified into global headlines and ~$200 million in investment). Nearly 100 submissions from both major AI labs (e.g., Anthropic, OpenAI) and smaller open-source projects (e.g., TRAE, Refact.ai) compete on its leaderboard, where scores are treated as indicators of real-world coding ability.

1.2 Approaches to SWE-bench Verified

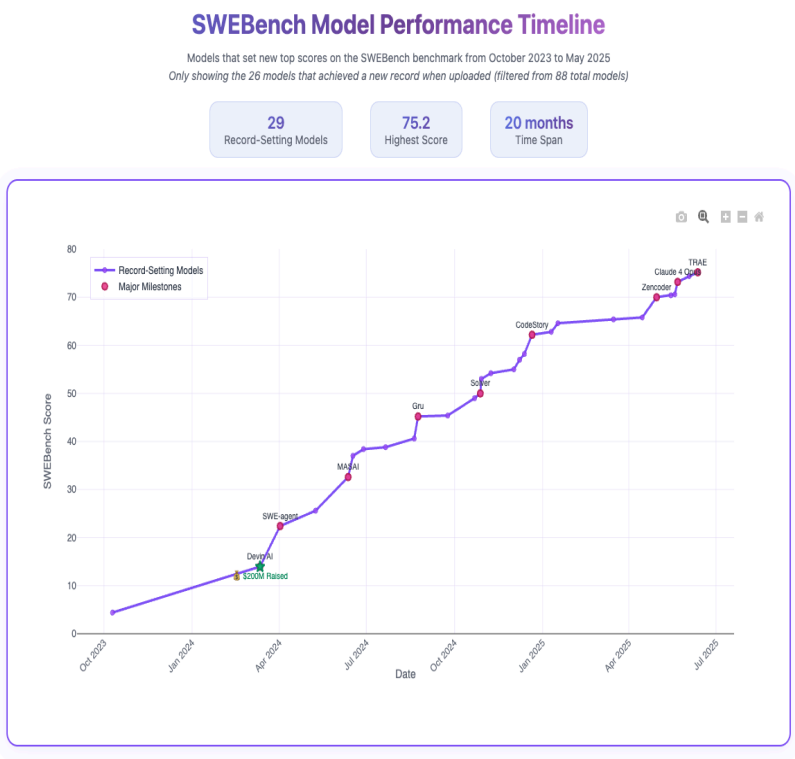
A line of recent systems targets SWE-bench Verified with different levers: data scaling, agent scaffolding, and stronger base models. A few examples:

- SWE-Dev focuses on both training and inference scaling, and reports that their 7B and 32B models can achieve top performance among all open SWE agents, reaching 23.4% and 36.6% pass@1, respectively (paper, code).
- SWE-smith introduces a large-scale data generation pipeline (~50k instances across 128 repositories) and trains SWE-agent-LM-32B to 40.2% pass@1 on SWE-bench Verified (paper, code).
- mini-SWE-agent presents a compact, ~100-line Python agent-scaffold and claims up to 68% pass rate on SWE-bench Verified, illustrating how lightweight orchestration and prompt design can materially lift scores (code).

- The OpenHands platform (formerly OpenDevin) provides an open, modular framework for building and evaluating development agents, command-line interaction, browser / tool use, and benchmark harnesses; which many subsequent evaluations build upon (paper, OpenReview).
- Commercial and production-oriented systems have also reported strong results. E.g., Anthropic notes Claude 3.5 Sonnet at 49% pass@1 success rate on SWE-bench Verified.
- More recently, Warp announced a 71% score on SWE-bench Verified. Unlike many leaderboard entries dependent on heavy scaffolding, Warp's evaluation differed only minimally from its user-facing product. The company explored multi-agent strategies and best-of-k patch generation, but ultimately their strongest configuration was the single-agent architecture offered to developers with only "minimal changes", making their score more representative than typical benchmark-only systems.

2. SWE-bench Performance Trends

While the early days of SWE-bench Verified benchmarks saw explosive progress where each new success represented a significant leap in AI coding abilities, the recent gains have been incremental at best despite fierce competition among the world's premier AI labs and a significant investment in reasoning model improvements. Therefore, focus has shifted from advancing the state of the art to optimizing performance for the benchmark.



As such, recent benchmark scores are less of a barometer of AI systems' competence in real world software engineering. Submissions frequently employ hidden scaffolding such as best-of-k patch generation, iterative refinement loops, and task-specific prompting – techniques explicitly noted in analyses of SWE-bench submissions as inflating leaderboard results without being reproducible in real-world deployment. The models

and scaffolds that secure high scores on SWE-bench Verified are rarely the same systems that practitioners deploy in production.

This is indicative of two problems, both of which limit the benchmark's efficacy as a predictor of real-world utility and hinders progress toward trustworthy, autonomous software development.

2.1 Benchmark Gaming

SWE-bench Verified is intended to be a reliable tracker of progress in autonomous software development. However, recent succeses have been more a reflection of finding incremental gains through overfitting the problem space with various scaffolding techniques. While scaffolding can meaningfully improve performance, it also introduces components that are typically absent at deployment, creating a gap between benchmark scores and user-facing behavior.

For example, analysis in SWE-Bench+: Enhanced Coding Benchmark for LLMs found that 32.67% of passing patches were due to solution leakage and 31.08% were due to weak test coverage; after filtering these, SWE-Agent + GPT-4's pass@1 dropped from 12.47% to just 3.97%. Similarly, evaluations on SWE-bench-Live – a dynamic benchmark designed to mitigate contamination and overfitting – showed that the same agent–model pairs performed substantially worse compared to their SWE-bench Verified scores, highlighting how static benchmarks may exaggerate true performance.

2.2 SWE-bench Verified Limitations

Our analysis yields that 74 of the 500 (~15%) of the problems in this benchmark had not been solved by any AI system until now; and while Blitzy managed to resolve 30 of them, the remaining 44 continue to be out of reach. We theorise that this is not because AI progress is stalling; but because the problem statements are insufficient and do not represent best prompting practices (or in some cases actively mislead the AI models).

The fact that recent score improvements have been fractional is indicative that AI models are already hitting the wall against these seemingly intractable but non-AI issues. It is unlikely that solving these remaining problems is a productive challenge for AI systems in the coming months when they are capable of much more, as demonstrated by Blitzy. Therefore we believe that we are approaching the limits of SWE-bench Verified's ability to track progress in AI software generation.

3. The Blitzy Platform

Blitzy is a full-stack autonomous software development platform designed for enterprise environments. We used Blitzy in its standard deployed form without any additional scaffolding or task-specific refinements on SWE-bench Verified, and achieved a staggering Pass@1 success rate of 86.8*%; which is to-date the largest single increase (*13%) over the previous best score (in contrast, the average first place jump is only 2.59%, and the past 11 submissions have only averaged a 1.33% increment). Our SWE-bench Verified evaluation submission can be found here.

Thus Blitzy's high score represents the first meaningful performance leap since May 2025; and re-establishes SWE-bench Verified as a beacon of real innovation rather than benchmark gaming. The architectural breakthroughs that propel Blitzy's success are designed to produce higher-quality code for enterprise-grade development projects using a novel approach to autonomous software generation, and not to optimize for benchmarking.

We intend to focus on the following aspects of our SWE-bench Verified benchmarking:

1. Blitzy's system 2 AI multi-agent architecture that enables effective understanding of multi-million line codebases at ingestion, reasoning across millions of lines of code, and alignment of local edits with global repository requirements.
2. A reproducible evaluation pipeline that transforms benchmark instances into structured repositories, executes solutions under strict pass@1 conditions, and logs results transparently.
3. Empirical analysis of the remaining reportedly "unsolvable" instances and the resulting benchmark ceilings, which highlight the limitations of SWE-bench Verified as a measure of progress.
4. Recommendations for future, representative benchmarks that move beyond localized tasks to capture repository-scale distributed reasoning, maintainability, and long-term software health.

3.1 Blitzy Architecture and Workflow

System 2 AI

System 1 vs System 2 thinking was popularized by Daniel Kahneman, and applied in AI contexts by Gary Marcus and Ernest Davis (NYU) in their paper *Thinking Fast and Slow in AI* and expanded upon in *System 2 Reasoning in Artificial Intelligence*. Both of these papers were published before the release of GPT-3, but the ideas remain relevant. They argued that most AI excels at System 1 pattern recognition while lacking System 2 deliberate reasoning for planning and abstraction. This more deliberate reasoning has

been applied at the model layer, and narrowly at the application layer, but Blitzy is pushing the frontier of this methodology.

Code Ingestion to Enable Domain-specific (Enterprise Software Development) Context Engineering

Blitzy performs an ingestion phase in which the entire source code corpus is transformed into a hierarchically summarized, relational index. Each line of code is embedded semantically while preserving the full set of relational dependencies inherent in software systems, including control-flow, call graphs, inheritance hierarchies, and module dependencies. Because software is intrinsically relational, these relationships are captured exhaustively and modeled as a structured graph that spans all levels of abstraction, from statements to modules and services. This relational structure is designed to be language-agnostic, enabling consistent representation across heterogeneous codebases.

Regardless of programming language, the ingestion process normalizes relationships into a common intermediate schema, ensuring that polymorphic or cross-language dependencies (e.g., Python calling into C extensions, or COBOL systems interacting with Java services) are preserved.

During inference, this index underpins a just-in-time, ranking-based context management system. Relevant fragments are retrieved and ranked based on both semantic similarity and relational proximity, allowing the model to surface only the most contextually useful information. This effectively extends the usable context without bound in real world software systems, enabling just in time context injection for every line of a codebase.

Extended Inference Time Validation

Blitzy's proprietary multi-agent, multi-model architecture leverages a combination of the best in class generative AI models (e.g. Anthropic, OpenAI, Google, etc.) to optimize the accuracy and performance of specific tasks, by using the most suitable model for the relevant purpose (e.g. Blitzy may use one LLM for code generation, and another for pre-compilation or validation).

Blitzy treats each software development task as an independent project. A project can be initiated for greenfield development or to enhance an existing codebase. These enhancements can range from new feature development and bug fixes to security improvements and documentation generation.

Blitzy's architecture is intended to optimize for code quality, not costs. Every time a file needs to be edited in a codebase, the platform will generate ad hoc tests, run tests, write the change, recompile code, then re-run the ad hoc unit test. Additionally, at the end of all

changes, the platform re-compiles and runs existing tests contained within the source code. Deviations from 'pass to pass' on any test triggers a validation workflow to dynamically understand and address the deviation.

With this agentic spec driven and test driven development process, Blitzy can effectively write millions of lines of code without straying from the original intent or executing un-intended changes.

For SWE-bench Verified benchmarks, which provide smaller problems, we used the Blitzy platform's "Fix Bugs" feature, which not only generates code to address the specified issue but also autonomously craft and run relevant tests as described above. This allows it to appropriately localize the bug fix while managing its downstream effects and preserving valid existing behaviors in order to avoid regressions.

The benchmark workflow is listed below:

1. Fork the Github repos corresponding to the 12 codebases in the benchmark to optimize local work
2. For each SWE-bench Verified issue, create a branch in the relevant repo corresponding to the commit from when the issue was reported
3. Create a project in the Blitzy platform for each issue
4. In each project, provide the corresponding branch URL to Blitzy for ingesting the codebase as it was at the time of the issue reporting
5. Blitzy's specialized army of code ingestion agents create an internal mapping of the code in Blitzy and a corresponding technical specification documenting Blitzy's understanding of the codebase as it was at that time
6. Prompt the Blitzy project with only the problem statement for the issue and initiate the bug fix, which updates the technical specification with Blitzy's understanding of the problem description and the potential fix(es) its AI agents have thought of
7. Blitzy's code generation agents complete code changes associated with the bug fix and create an associated pull request against the input branch, as well as a project guide documenting how the fix was implemented and verified
8. A patch file generated from the pull request was used to evaluate whether the bug fix was successful (i.e. whether the result was pass or fail) using both sb-cli and modal for cloud-based scoring as prescribed in the SWE-bench Verified documentation

As Blitzy is designed to focus on real-world application for autonomous software development and committed to reproducibility of its outcomes, the platform enables web searching by default for its coding agents. This enables Blitzy to gather additional relevant knowledge about the problem it is addressing and leads towards better quality code generation.

However, for an open-source benchmarking task such as SWE-bench Verified, this also poses a risk of finding records of solutions or helpful hints and gain unfair advantage towards solving the problem.

To eliminate such contamination risks, for the SWE-bench Verified runs Blitzy explicitly blacklisted the relevant project repositories and online resources (e.g. Google Groups) that referred to leaked or publicly indexed solutions, hints, or tests; preventing our AI agents from retrieving any code or other material that could compromise fair scoring. This was double-checked manually by inspecting the LLM traces from the AI models used by Blitzy and confirming that there was no accidental leakage.

3.2 Transparency of Blitzy Results

Often, SWE-bench Verified benchmark leaders employ various scaffolding techniques (e.g., additional infrastructure layers that coordinate generation, error handling, prompt enhancements, or multiple attempts) to bolster scores. This produces unrealistic performance metrics that do not reflect real-world AI-assisted software development, and therefore detracts from the intent of such benchmarks.

While the Blitzy platform is proprietary, its out of the box usage in its standard deployed form without any scaffolding agents means that any customer of Blitzy can follow the above procedure and replicate our benchmark results. Since Blitzy mirrors the conditions under which enterprises expect reliable outputs, we bridge a critical gap between leaderboard results and production utility – where some other systems may optimize for controlled evaluations, Blitzy's architecture is designed for the unpredictability of real-world repositories.

Similarly, Blitzy performs single-shot code generation instead of parallelly generating multiple code patches for fixing the issue and then picking the best one (i.e. pass@k solutions). Blitzy does not enhance the problem statement prompts either, staying true to the original bug report.

3.3 "Unsolvable" Instances

An analysis of the previous top 26 SWE-bench Verified submissions reveals a persistent ceiling: 74 of the 500 task instances have never been solved, and another 23 have been solved by only one or two models. Together, nearly 15-20% of the benchmark is effectively intractable under current conditions, placing the theoretical maximum score at roughly 85% even if every other task were solved.

The source of this difficulty lies not in model capability but in the problem statements themselves. Many are vague, inaccurately describe the underlying bug, or are cluttered

with extraneous and sometimes misleading information. As a result, the unsolved tasks expose dataset flaws rather than reasoning limits. UTBoost claims that addressing the test deficiencies of 26 such issues affected the leaderboard rankings for 24% agents (paper, code).

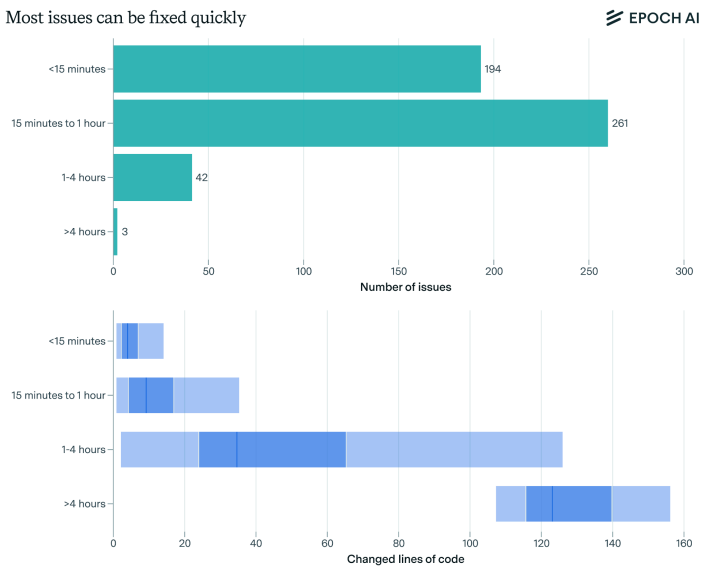
While Blitzy managed to solve 30 such issues for the first ever time, it is doubtful that all 74 instances will ever be solved by any AI system. More importantly, pursuing them is not a productive challenge – high-quality benchmarks should test models under best prompting practices to assess their true capabilities.

For system 2 AI architectures like Blitzy, progress should be measured not by eking out marginal gains on ambiguous bug reports, but by tackling repository-scale improvements and refactors. Just as SWE-bench Verified supplanted SWE-bench Full one year after its release, the field now stands at another inflection point; where a new benchmark is needed to better encapsulate the evolution of autonomous software engineering.

3.4 Design Limitations of SWE-bench Verified

The SWE-bench Verified benchmark, while including real-world issues, is not truly representative of modern, large-scale software development that happens at major enterprises. Its narrow focus on isolated, single-issue problem solving skews the nature of code generation to quick fixes.

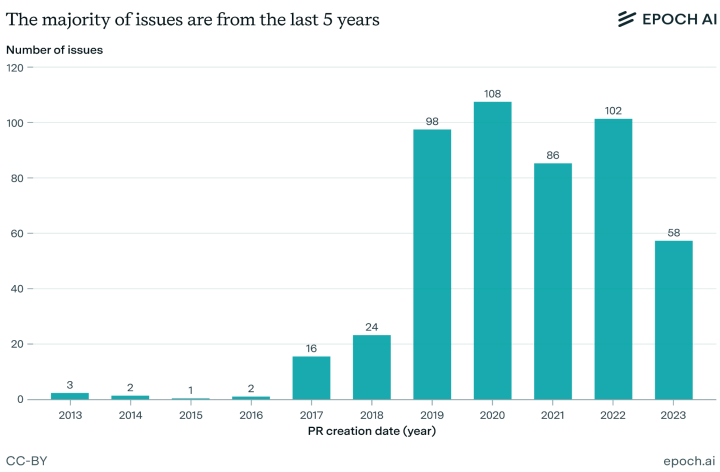
For instance, an analysis of SWE-bench Verified by Amazon finds that 78% of changes only touch functions, not classes, with 1.87 functions needing to be changed on average. Furthermore, It contains only Python repositories, all the tasks are bug fixes, and the Django repository is significantly over-represented at over 45% of all tasks.



Similarly, a study by Epoch found that based on SWE-bench Verified creators' estimates of estimates for how long it would take "experienced software engineer familiar with the codebase" to resolve the issues, such quick fixes (taking <15 min) average only 5 changed lines of code (LOC), while tasks in the '15 min - 1 hour' bucket average 14 changed LOC. Together, these two categories make up 91% of the SWE-bench Verified problem set; making it only relevant to

real world coding as it pertains to micro bug fixes.

Along with the limitations of the SWE-Bench architecture, the task instances themselves are also limited. The Epoch study raises concerns about its relevance to current software development practices based on the age of the problems, citing that half of the issues are from before 2020 with some dating back more than 12 years. Many of the packages and versions used then are already end of life (EOL), or will be soon.



Another concern about a static benchmark like SWE-bench Verified is that new LLMs are likely to overfit on the repositories including their histories and even the issues themselves; as these public repositories are an important source of training data. To counter these effects, future benchmarks could diversify beyond Python, weigh toward multi-function changes, and avoid EOL repos.

4. Conclusion

We introduced Blitzy, a multi-model, multi-agentic system 2 AI platform designed for enterprise-scale, enterprise grade software generation. Through unique innovations like domain specific context engineering and collaborative agents that maintain global context for local code authoring; Blitzy achieves a paradigm shift in AI-assisted software development; and demonstrates that agent specialization, inference-time validation and clever orchestration yield more robust systems than a standalone general-purpose model.

This is highlighted by its significant leap forward on SWE-bench Verified with an out of the box Pass@1 score of 86.8%, not to mention that it managed to solve 30 issues that had never succeeded in any of the past 26 winning entries.

Blitzy's performance demonstrates that sophisticated reasoning architectures can overcome traditional limitations of AI models without relying on benchmark-specific optimizations or artificial performance enhancement techniques. The platform's ability to achieve high scores through genuine problem-solving capabilities rather than scaffolding or multiple-attempt strategies establishes a new standard for what AI systems can accomplish in complex development environments.

We call for a new, dynamic, comprehensive, and representative software engineering benchmark that better reflects the real-world, large-scale code generation tasks regularly encountered by human developers; where success can easily translate to actual productivity improvements for the software development community and the organisations that employ them. The path forward requires collaboration between AI researchers, software engineering practitioners, and evaluation methodology experts to transform how AI coding systems are evaluated.

References

- Allen Institute for AI. (n.d.). SWE-bench. GitHub. https://github.com/allenai/SWE-bench
- Claude AI. (2025). SWE-bench Autonomous research paper. Claude Artifacts. https://claude.ai/artifacts/a73f1c3a-d587-49dd-b725-636d68415036
- Denain, J.-S. (2025, June 13). What skills does SWE-bench Verified evaluate? Epoch AI. https://epoch.ai/blog/what-skills-does-swe-bench-verified-evaluate
- Hugging Face. (n.d.). SWE-bench. Hugging Face. https://huggingface.co/SWE-bench
- Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Marcus, G., & Davis, E. (2020). *Thinking Fast and Slow in AI*. *arXiv preprint* arXiv:2010.06002. https://arxiv.org/abs/2010.06002
- Marcus, G., & Davis, E. (2021). *System 2 Reasoning in Artificial Intelligence*. *arXiv preprint* arXiv:2110.01834. https://arxiv.org/abs/2110.01834
- METR. (2025, July 10). Measuring the impact of early-2025 AI on experienced open-source developer productivity. METR Blog. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- OpenAI. (2024, March 5). Introducing SWE-bench Verified. OpenAI. https://openai.com/index/introducing-SWE-bench-verified/
- Stanford NLP. (n.d.). DSPy SWE-bench examples. GitHub. https://github.com/stanford-nlp/dspy/tree/main/examples/SWE-bench
- SWE-bench. (n.d.). Experiments. GitHub. https://github.com/SWE-bench/experiments
- SWE-bench. (n.d.). sb-cli. GitHub. https://github.com/SWE-bench/sb-cli
- SWE-bench. (n.d.). sb-cli: API authentication. GitHub. https://github.com/SWE-bench/sb-cli#api-authentication
- SWE-bench. (n.d.). sb-cli: Example prediction format. GitHub. https://github.com/SWE-bench/sb-cli#example-prediction-format
- SWE-bench. (n.d.). sb-cli: Usage. GitHub. https://github.com/SWE-bench/sb-cli#usage
- SWE-bench. (n.d.). SWE-bench official website. https://swebench.com

- Wang, L., Mishra, S., Zhao, S., et al. (2023). SWE-bench: Can language models resolve real-world GitHub issues? arXiv. https://arxiv.org/abs/2310.06770
- Wang, Y., Pradel, M., & Liu, Z. (2025). Are "solved issues" in SWE-bench really solved correctly? An empirical study. arXiv preprint. https://arxiv.org/html/2503.15223v1