
Writing a Research Paper with an AI Agent: A Chronicle of KISS Sorcar Writing Its Own Paper

Koushik Sen

EECS Department, UC Berkeley
ksen@berkeley.edu

Abstract

We describe, step by step, how KISS Sorcar—an AI coding assistant built on the KISS Agent Framework—was used to write its own research paper. The process spanned nine days and comprised over one hundred user-issued tasks, each recorded in a SQLite database alongside the full git history of every resulting change. The record reveals a distinctive human–AI collaboration pattern: the human author steered the paper at the level of intent (“add a section on evaluation,” “switch to active voice,” “cite recent papers”), while the agent executed each directive—editing $\text{\LaTeX}$, searching the web for citations, compiling with $\text{\pdflatex}$, checking spelling, and fixing its own bugs when the diff/merge interface failed. The entire process also doubled as a stress test of the tool itself, since every agent-introduced defect immediately impaired the agent’s own ability to work. We present the complete chronological record, organized into seven phases, and draw lessons about the practical dynamics of AI-assisted academic writing.

1 Introduction

KISS Sorcar is an open-source AI assistant and integrated development environment (IDE) built on the KISS Agent Framework Sen [2026]. It operates inside Visual Studio Code, accepts natural-language tasks, and executes them through a layered agent hierarchy that manages budget tracking, automatic continuation, coding tools, persistent chat, and git worktree isolation.

The first author used KISS Sorcar to write the research paper that describes KISS Sorcar itself. Every task the user issued was recorded in a SQLite database (`sorcar.db`), and every change the agent made was committed to git. This paper reconstructs that process from the raw record.

The record shows that the human author never typed $\text{\LaTeX}$ from scratch. Instead, the author directed the agent through short, imperative sentences: compile the paper, add related work, switch to active voice, cite a specific URL, check spelling. The agent, in turn, read source files, searched the web, edited $\text{\LaTeX}$, ran $\text{\pdflatex}$ and $\text{\bibtex}$, and presented diffs for the author to accept or reject. When the agent introduced bugs—as it did on at least two occasions—the author’s next task was to fix those bugs, and the agent fixed them by writing integration tests first.

We organize the chronology into seven phases and present every substantive task the user issued. We omit only exact duplicates (tasks re-submitted because the first attempt timed out or failed).

2 Background: How KISS Sorcar Executes a Task

When the user types a task into the KISS Sorcar chat panel, the following sequence occurs:

1. The **Worktree Sorcar Agent** creates a git worktree on a fresh branch, isolating the work from the main branch.

2. The **Chat Sorcar Agent** loads prior conversation context so the agent can refer to earlier tasks.
3. The **Sorcar Agent** selects tools: file reading, file editing, bash commands, web browsing, and parallel sub-agents.
4. The **Relentless Agent** manages context length by summarizing completed sub-sessions and continuing across boundaries.
5. The **KISS Agent** executes a budget-tracked ReAct loop, calling the LLM and tools in alternation until the task is done or the budget is exhausted.

Every file the agent modifies is presented in a diff/merge UI. The user can accept or reject each change. If the user has enabled auto-commit, accepted changes are committed to git automatically.

3 Chronology

The paper-writing process spanned nine calendar days. We divide it into seven phases.

3.1 Phase 1: Scaffolding and First Compilation

The user started writing the paper by giving KISS Sorcar the following task:

“Can you write a paper on KISS Sorcar using the NeurIPS 2026 format? You MUST read PWD/README.md and the project source code to write the paper.”

The paper’s $\text{\LaTeX}$ source was committed to the repository as part of a squashed commit that added the NeurIPS 2026 submission template, a checklist file, and a 612-line draft of `kiss_sorcar.tex`.

A second squashed commit added formatted prompt blocks—converting the inline prose descriptions of agent prompts into explicitly formatted Verbatim environments using the `fancyvrb` package. This commit added roughly 300 lines.

The user then executed the task:

“Can you compile the paper PWD/papers/kiss_sorcar.tex using pdflatex? I just installed mactex in the system.”

Three minutes later, the user asked the agent to explain each instruction in the system prompt:

“Can you show each instruction in the SYSTEM.md separately and explain its rationale in the paper?”

The agent read `src/kiss/SYSTEM.md`, decomposed it into individual instructions, and wrote a rationale for each one in the $\text{\LaTeX}$ source.

Two tasks followed in quick succession, both demanding a thorough related-work section:

“Can you add more related work to the paper? Use internet search extensively. Validate the existence of citations.”

“You MUST discuss related work and cite recent papers published after August 2025. Make sure that the papers have high citation counts. Read the papers to judge relevance.”

The agent searched the web, retrieved papers, verified their existence, and added citations. The resulting commits show the related-work section expanding with comprehensive literature coverage.

3.2 Phase 2: Structural Rewriting

The next morning brought a burst of structural changes. The user pointed that the implementation had changed and told the agent:

“Note that the implementation has changed, specifically that of `relentless_agent.py`. Update the paper accordingly.”

The agent read the updated source code and revised the paper to match. The user then asked for detailed explanations of each system prompt instruction (task 75) and asked the agent to incorporate the latest commit (task 77).

The user issued two critical style directives:

“Instead of writing the paper in the passive voice, write it as ‘we’.”

This produced a commit titled “docs: refactor paper to use active voice with ‘we’ as subject.” The user also renamed “Stateful Sorcar Agent” to “Chat Sorcar Agent.”

The user then asked for a new section on the VS Code extension’s unique features, instructing the agent to search the internet for comparison:

“Can you add a section in the paper on the VS Code extension named KISS Sorcar about its unique features that are not present in other IDEs or assistants?”

The agent searched the web, compared features with Copilot, Cursor, and Devin, and wrote Section 5 of the paper.

After the user updated the system prompt, the agent was told to synchronize:

“I have updated the system prompt significantly. Please update the paper and build it.”

3.3 Phase 3: Evaluation and Metrics

The afternoon session focused on evidence. The user asked the agent to cite sources precisely (task 96) and to add a section on Terminal Bench 2.0 using preliminary results from the jobs/directory (task 97). The agent wrote the evaluation section and computed aggregate results from the JSON result files.

The user asked the agent to write the introduction properly with citations (task 98), then to add lines-of-code counts for the four agent classes (tasks 100–101). After these additions, the user stepped back and asked for coherence:

“Can you make the sections coherent in the paper?”

Bug discovery. Task 104 revealed a bug in the tool itself: the agent had modified the paper and auto-committed the changes, but the commit never appeared in the report. The user’s response was characteristic:

“Your last task showed a bug. The result said the file has been modified and the user auto committed the changes, but that commit never showed up in the report. You MUST write a test...”

The agent wrote an integration test, reproduced the bug, and fixed it. The paper-writing session doubled as a stress test.

3.4 Phase 4: Citations, Formatting, and File Organization

The next day was the most active. It began with a second bug fix: the diff/merge UI had not appeared when the agent edited the paper. The user wrote:

“Look at the last task. You updated `kiss_sorcar.tex`, but the diff/merge UI didn’t show up. Reproduce this bug by writing an integration test and fix.”

The agent wrote the test, found the defect, and fixed it (task 109).

The user then updated the evaluation with the latest five runs (task 112), cross-checked the Cursor Composer 2 score using internet search (task 113), and added citations for Claude Code, OpenAI Codex, and OpenClaw (task 115).

After several build attempts, the user cleaned up the paper’s structure:

- Cited the “cheating agents” blog and its corresponding paper (task 119).
- Removed the NeurIPS checklist (task 122).
- Wrote the abstract from the introduction’s key points (task 123).
- Created a BibTeX bibliography file (task 125).
- Moved all paper files into a `kiissorcar/` subdirectory (task 126).
- Fixed `.gitignore` rules for L^AT_EX build artifacts (task 127).
- Updated the conclusion with evaluation results (task 128).
- Included a screenshot of the KISS Sorcar UI (task 131).
- Added citations for GEPA, AlphaEvolve, and OpenEvolve (task 136).

The user then added a case study section drawn from a backed up history database, showing how KISS Sorcar had been used to modify the worktree feature’s workflow through natural-language commands (tasks 140, 142). This section demonstrated “painless software engineering”—the user described a desired workflow change in English, and the agent implemented it.

The afternoon brought a long spell-checking session. Four consecutive attempts (tasks 147–150) to check spelling suggest the agent struggled to complete the task within budget; the fifth attempt (task 151) specifically instructed: “Fix the 4 typos found.” The agent then compiled the paper to verify (task 152).

The user updated the system prompt again and asked the agent to synchronize the paper (task 167), then to discover all inconsistencies (task 169). Two tasks checked citations for AI slop—fabricated references that LLMs sometimes invent (tasks 170–171).

Between agent sessions, the user edited the L^AT_EX source directly on two occasions. Both episodes illuminate the boundary between what is faster to type and what is faster to delegate.

The first manual edit was a paragraph insertion. The user had just reviewed the “Painless Software Engineering” case study section (Section 7, tasks 140, 142), which described a real development session in which the worktree merge workflow was first understood and then redesigned through conversational prompts. The user realized that the section lacked a framing observation: the underlying pattern—ask the agent to generate a detailed, step-by-step description of a buggy workflow, then ask it to fix the buggy steps in natural language—was general and worth stating explicitly. Rather than formulate a prompt to describe this edit, the user opened the file and typed seven lines directly into line 1399. Git captured the insertion in a “baseline from dirty state” commit. The manually typed paragraph contained a minor grammar error (“ask KISS Sorcar change some buggy step” instead of “ask KISS Sorcar *to* change some buggy step”), which the agent later corrected during a grammar-check pass (task 183).

The second manual edit was four spelling corrections, and it arose from frustration. Tasks 147–150 were four consecutive attempts to spell-check the paper; each either timed out or exceeded its budget before completing the fix. After the fourth failure, the user opened the file and corrected the four typos directly: “Assitant” → “Assistant” in the title, “Farmework” → “Framework” in the architecture section, and “hish” → “high” and “comapred” → “compared” in the evaluation discussion. All four errors had been introduced by the agent in earlier editing sessions. When the next agent session began thirty minutes later, the worktree isolation mechanism detected the uncommitted changes and created a baseline commit. Task 151 (“Fix the 4 typos found”) then produced an empty diff: the user had already applied exactly the changes the agent would have made.

The “baseline from dirty state” mechanism deserves a brief note. Whenever a new agent session starts, KISS Sorcar checks whether the working tree contains uncommitted changes. If it does, the system creates an automatic baseline commit before the agent begins its own work. This means the user can edit any file at any time—between tasks, during a coffee break, even while the agent is running on a different worktree—and the edits will be preserved in git without the user needing to commit manually. The mechanism enabled the seamless coexistence of manual and agent-mediated editing observed here.

After the manual edits, the agent checked grammar and spelling (tasks 182–183) and built the paper (tasks 185–186).

3.5 Phase 5: Refinement and Citation Hygiene

The user built the paper, reviewed the generated PDF across all 23 pages (task 194), and asked the agent to cite specific frontier models—RLM 5.1, KIMI K2.5, Composer 2.0, Anthropic Opus 4.6—and LiveCodeBench (tasks 198–199).

The next day, the user reformatted the paper so that each paragraph occupied a single line (task 224) and configured VS Code to wrap `.tex` files (task 225). Multiple rounds of system prompt synchronization followed (tasks 226–227, 233, 261, 274), each triggered by the user updating `SYSTEM.md` and asking the agent to propagate the changes.

The user asked for syntax-highlighted code listings (tasks 245–248). The agent first used the `minted` package, which requires `-shell-escape` and the `pygments` Python library. When the PDF showed `<MINTED>` placeholders instead of highlighted code (task 257), the user directed the agent to abandon `minted` and use `listings` with a `verbatim` fallback (task 264). The agent replaced `minted` with `lstnewenvironment`.

The user then reported 74 undefined citations (task 265) and asked the agent to validate all citations using internet search (task 266). The agent corrected paper years, fixed repository URLs, and escaped underscores in $\LaTeX$ prose.

Finally, the user asked the agent to read the paper and check for inconsistencies (task 270).

3.6 Phase 6: System Prompt Updates and the Website

The user built the paper again (task 302) and ran GEPA experiments on HotpotQA (task 293)—work related to the broader project but not the paper itself.

The next day, the user updated the system prompt’s Sorcar-specific instructions and asked the agent to synchronize the paper and build it (task 312). The user then shifted to outreach:

“Can you create a nice, simple, elegant website for KISS Sorcar at `kissorcar.github.io` and upload it?”

The agent created the website, and the user iterated on its design over eight tasks (320–342)—adjusting the background color, logo size, layout, content order, and styling. The user asked the agent to add the website link and paper citation to `README.md` (task 340) and to emphasize disciplined engineering practices on both the website and the `README` (task 342).

3.7 Phase 7: Final Polish and Lecture Slides

In the final sessions, the user integrated a message into the paper’s abstract, introduction, and conclusion (task 418) and added all authors from `README.md` to the website (task 420).

The user then asked the agent to prepare PowerPoint lecture slides based on the paper:

“I am going to give an hour-long lecture on KISS Sorcar and the KISS Agent Framework. Can you prepare PowerPoint slides for me based on the paper?”

The agent generated the slides using `python-pptx`. The user refined them over several tasks—fixing code formatting on specific slides (tasks 436, 439, 440) and adding a features slide drawn from the website (task 437).

By the ninth day, the paper had reached its final form: 893 lines of $\LaTeX$, a 23-page PDF, and a companion website at `kissorcar.github.io`.

4 Quantitative Summary

Table 1 summarizes the paper-writing process.

5 Patterns of Collaboration

The record reveals several recurring patterns in the human–AI collaboration.

Table 1: Quantitative summary of the paper-writing process.

Metric	Count
Calendar days	9
Distinct paper-related user tasks	~100
Git commits touching the paper	82
Final paper length (lines of L ^A T _E X)	893
Final PDF length (pages)	23
“Build the paper” tasks	25
Spelling/grammar check tasks	8
System prompt synchronization tasks	9
Citation-related tasks	8
Bugs discovered during writing	2

The human steers; the agent rows. The user never typed L^AT_EX from scratch. Every structural decision—add a section, switch to active voice, use Strunk and White style, include a screenshot—came from the user. The agent executed each directive, often reading source code, searching the web, or running shell commands to gather the information needed.

Build–inspect–fix cycles. Twenty-five of the roughly one hundred tasks were simply “build the paper.” These were not wasted effort; each build let the user inspect the PDF and issue the next directive. The cycle resembles the edit–compile–run loop of traditional programming, compressed by the agent’s ability to make substantive changes between builds.

System prompt co-evolution. Nine tasks asked the agent to update the paper after the user had changed SYSTEM.md. The paper describes the system prompt in detail; every time the prompt changed, the paper had to change too. The user enforced this invariant manually, but the agent performed the actual synchronization.

Citation hygiene as a distinct phase. The user devoted multiple tasks to verifying citations: checking for AI slop (fabricated references), validating existence via internet search, correcting years and URLs, and adding missing references. This suggests that LLM-generated citations require a dedicated verification pass, not merely a final proofread.

Bug discovery as a side effect. Two bugs in KISS Sorcar itself were discovered during the paper-writing process. In both cases, the user asked the agent to reproduce the bug with an integration test and then fix it. The paper-writing task thus served as a continuous stress test of the tool.

Occasional direct editing and bidirectional polishing. Although the user delegated nearly all L^AT_EX editing to the agent, the git history reveals two occasions on which the user edited the source file by hand: once to insert a paragraph describing a workflow insight and once to fix four spelling errors after four failed agent attempts. Both edits were small, targeted, and faster to type than to describe in a prompt. The “baseline from dirty state” mechanism—an automatic commit of any uncommitted changes at the start of each agent session—preserved these edits without requiring the user to interact with git.

What makes the pattern noteworthy is that it ran in both directions. The agent introduced the four spelling errors in earlier editing sessions; the user fixed them by hand. The user’s manually typed paragraph contained a missing word; the agent fixed it during a subsequent grammar-check task. Each party cleaned up after the other. This bidirectional polishing suggests that the most productive mode of human–AI collaboration is not one in which the agent produces perfect output, but one in which both parties iterate toward correctness, each catching errors the other introduced. The git history preserves the full trace of these corrections, making the collaboration auditable after the fact.

Retry and refinement. Several tasks were submitted multiple times—spelling checks four times, builds two or three times in succession. Some retries were caused by timeouts or budget exhaustion; others by the user’s dissatisfaction with the result. The agent’s stateless-per-task design meant that each retry was independent.

Formatting struggles. The attempt to use the `minted` package for syntax highlighting consumed five tasks and three commits before the user gave up and switched to `listings`. $\text{\LaTeX}$ package compatibility remains a friction point even for an agent that can run shell commands.

6 Lessons Learned

1. **Short, imperative tasks work best.** The most effective tasks were one or two sentences long and stated a clear objective. Vague tasks (“make the sections coherent”) produced acceptable but less predictable results.
2. **Verification is the human’s job.** The agent can build the paper and check spelling, but only the human can judge whether the argument is sound, the emphasis is correct, and the citations are honest.
3. **Co-evolution creates maintenance burden.** When the paper describes its own system prompt, every prompt change triggers a paper change. An automated consistency check—or a single source of truth—would reduce this burden.
4. **Citation verification must be explicit.** LLMs sometimes fabricate references. The user’s explicit “check for AI slop” tasks caught these errors, but a less vigilant user might not.
5. **The tool tests itself.** Using KISS Sorcar to write its own paper created a feedback loop: bugs in the tool impaired the paper-writing process, which made the bugs visible, which led to immediate fixes. This self-hosting discipline is a powerful quality signal.

7 Complete Task Log

Table 2 lists every paper-related task the user issued, in chronological order. Tasks marked with an asterisk (*) were retries of a failed or timed-out attempt.

Table 2: Complete chronological log of paper-related tasks.

ID	Task (abbreviated)
48	Write a paper on KISS Sorcar using the NeurIPS 2026 format; read README.md and source code.
49	Compile the paper using pdflatex (mactex just installed).
50	Show each SYSTEM.md instruction and explain its rationale in the paper.
51	Add more related work; use internet search; validate citations.
52	Discuss related work; cite recent papers after Aug 2025; verify citation counts.
74	Implementation of <code>relentless_agent.py</code> changed; update paper.
75	In Section 4, explain each instruction in detail with motivation.
77	Look at the last commit and update the paper.
78	Build the paper.
79	Switch from passive voice to “we.”
80	Build the paper.
82	Add section on VS Code extension’s unique features; search internet.
84	System prompt updated significantly; update paper and build.
96	Go over changes and cite sources precisely.
97	Add section on Terminal Bench 2.0; use preliminary results from jobs/.
98	Write the introduction nicely with proper citations.
100	Add lines of code (excluding comments and blanks) to paper.
101	Add LOC only for KissAgent, SorcarAgent, ChatSorcarAgent, WorktreeSorcarAgent.
102	Make the sections coherent.
104	Bug: auto-committed changes not in report. Write test and fix.
106	Rewrite paper following Strunk and White.
109	Bug: diff/merge UI didn’t appear. Reproduce with test and fix.
111	Build the paper.
112	Update Terminal Bench 2.0 evaluation with latest 5 runs.

Continued on next page

ID	Task (abbreviated)
113	Cross-check Cursor Composer 2 score; cite their arxiv paper.
115	Cite Claude Code, OpenAI Codex, and OpenClaw.
116	Build the paper.*
117	Build the paper.*
118	Move epigraph from preamble to after <code>\begin{document}</code> .
119	Cite cheating-agents blog and corresponding paper.
121	Build the paper.
122	Remove NeurIPS checklist and checklist.tex.
123	Write abstract from introduction's key points.
125	Create kiss_sorcar.bib and use BibTeX.
126	Move paper files to kissorcar/ subdirectory; update .gitignore.
127	Fix .gitignore rules for .aux, .out, .bbl, .blg.
128	Update conclusion with Terminal Bench 2.0 evaluation results.
131	Include screenshot of KISS Sorcar UI in paper.
136	Add citations for GEPA, AlphaEvolve, OpenEvolve.
140	Add case study from Dropbox history showing workflow modification.
142	Add "Painless Software Engineering" section from worktree history.
147	Check spelling.*
148	Check spelling.*
149	Check spelling.*
150	Check spelling.
151	Fix the 4 typos found.
152	Compile to verify typo fixes.
167	SYSTEM.md updated; update paper.
169	Discover all inconsistencies and fix them.
170	Check all citations for AI slop.
171	Check all citations for AI slop.*
173	Build the paper.*
174	Build the paper.*
175	Build the paper.*
176	Build the paper.
182	Check grammar and spelling of manual edits.
183	Check and fix grammar and spelling of manual edits.
185	Did you build the paper?
186	Build the paper.
192	Build the paper.
193	Build the paper.
194	Review PDF for formatting, layout, or citation issues (23 pages).
195	Build the paper.
197	Build the paper.
198	Cite RLM 5.1, KIMI K2.5, Composer 2.0, Anthropic Opus 4.6.
199	Cite LiveCodeBench.
205	Build the paper.
206	Build the paper.*
213	Build the paper.
215	Build the paper.
224	Reflow paragraphs to single lines; do not change content.
225	Configure VS Code to wrap .tex files.
226	Add discussion of SYSTEM.md hunk to paper.
227	Sorcar-Specific Overrides updated; update paper.
229	Build the paper.
233	Self-Improvement Loop changed in SYSTEM.md; update paper.
234	Build the paper.
242	Build the paper.*
243	Build the paper.
245	Include syntax-highlighted code for simple agent in Section 2.1.

Continued on next page

ID	Task (abbreviated)
246	Syntax-highlight all prompts.*
247	Syntax-highlight all prompts using coding package.*
248	Syntax-highlight all prompts; search internet.
249	Build the paper.
256	Remove line numbers from code listing.
257	PDF shows <MINTED> instead of prompts.
261	Update paper with updated Sorcar-specific instructions.
262	Build the paper.
264	Do not use minted; leave prompts verbatim.
265	Fix 74 undefined citations.
266	Check if all citations are valid; use internet.
270	Read paper and check for inconsistencies.
271	Build the paper.
274	Sorcar-specific instructions updated; update paper.
275	Build the paper.
302	Build the paper.
312	Sorcar-specific instructions changed; update paper and build.
418	Integrate message in abstract, intro, and conclusion; build.
432	Prepare PowerPoint lecture slides from paper.
436	Fix code formatting on slide 9.
437	Add features slide from website.
439	Fix code formatting on slide 9.*
440	Fix code formatting on slide 10.

8 Conclusion

The KISS Sorcar paper was written in nine days through a sustained dialogue between a human author and an AI agent. The human supplied intent and judgment; the agent supplied labor and speed. The process was not flawless—the agent fabricated citations, failed to render syntax highlighting, and introduced bugs in its own tool—but each failure was caught and corrected within the same workflow.

The complete record, preserved in three SQLite databases and 82 git commits, offers an unusually transparent view of AI-assisted academic writing. It shows that the collaboration works best when the human issues short, specific directives, verifies the output after each build, and treats citation hygiene as a first-class concern.

Writing a paper about a tool with the tool itself imposes an unusual discipline. Every defect in the agent becomes a defect in the paper-writing process, which the author notices and fixes. This self-hosting loop—building the tool, using it to describe itself, and fixing what breaks—may be the strongest argument for the tool’s reliability.

Acknowledgments

This research is supported in part by gifts from Accenture, Amazon, AMD, Anyscale, Broadcom, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Lightspeed, Mibura, NVIDIA, Samsung SDS, SAP, by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research through the X-STACK: Programming Environments for Scientific Computing program (DESC0021982,) and the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112590134.

Disclaimer: The paper has been edited in part using KISS Sorcar, an AI tool.

References

Koushik Sen. KISS sorcar: A stupidly-simple general-purpose and software engineering AI assistant, 2026. URL <https://arxiv.org/abs/2604.23822>.