Image-Based Development with LLM Agents: Building ai-theoretical.org in 72 Hours

Franco Cazzaniga[1] and Claude[2]

[1]Università dell'Insubria, `franco.cazzaniga@uninsubria.it`

[2]Anthropic

January 2026

Abstract

This paper documents the development of ai-theoretical.org, an automated academic preprint repository, built through direct collaboration between a human and an AI developer. After an initial setup phase ("Day 0") where the human configured the cloud infrastructure, approximately 72 hours of development followed with Claude (Anthropic) operating directly on a live Pharo Smalltalk image as the primary implementer. The project employed the Model Context Protocol (MCP) to give the language model full access to the runtime environment, enabling it to inspect objects, compile methods, execute tests, and iterate without human mediation at the code level. We describe the architecture, the development process, the distinctive nature of pair programming in this context, and the implications of this pattern for software development. The complete source code is available at `https://github.com/ai-theoretical/ai-theoretical.org`.

1 Introduction

The conventional pattern of using large language models for software development involves a cycle that, while useful, introduces friction at every iteration: the model generates code, a human copies it into an editor, executes it, observes the result, and reports errors back to the model. This loop places the human in the role of a proxy between the model and the runtime environment. The model cannot see what it is doing; it can only be told.

This paper reports on an experiment that eliminates this proxy role. By connecting Claude, Anthropic's language model, directly to a live Pharo Smalltalk image through the Model Context Protocol, we created an environment where the model could inspect, modify, and test code without human intervention at the implementation level. The human role shifted from code-level debugging to specification and acceptance.

The result was ai-theoretical.org, a fully functional preprint repository with automated submission handling, AI-assisted editorial review, static site generation, and deployment to GitHub Pages. After an initial day of infrastructure setup by the human author, the system went from initial Pharo code to production deployment in approximately 72 hours of Claude-driven development between December 31, 2025 and January 2, 2026.

We do not claim that this approach is universally superior to conventional development. We do claim that it represents a qualitatively different mode of human-AI collaboration—one that resembles pair programming but with an unusual division of labor—whose properties deserve investigation.

2 Background

2.1 The Model Context Protocol

The Model Context Protocol (MCP) is an open standard introduced by Anthropic in November 2024 [Anthropic, 2024]. It provides a uniform interface through which language models can connect to external tools and data sources. An MCP server exposes "resources" (readable data) and "tools" (executable actions); an MCP client—typically a language model host—can discover and invoke these capabilities through a standardized JSON-RPC interface [Model Context Protocol, 2025].

The significance of MCP for this work is that it allows the construction of bridges between language models and arbitrary computational environments. Rather than building custom integrations for each tool, a developer can implement an MCP server once and expose it to any MCP-compatible client.

2.2 Pharo and Image-Based Development

Pharo is a modern, open-source implementation of Smalltalk, a language family that pioneered object-oriented programming in the 1970s and 1980s [Goldberg and Robson, 1983]. Pharo descends from Squeak [Ingalls et al., 1997] and maintains the image-based development model that distinguishes Smalltalk from most contemporary languages.

In image-based development, the entire system state—classes, methods, objects, and their relationships—exists as a persistent memory image. There is no separate compilation step; modifying a method immediately changes the running system. The development tools (browser, debugger, inspector) operate directly on live objects. This architecture was designed for exploratory programming and has been shown to support rapid iteration [Ingalls et al., 2020, Black et al., 2009].

These properties make Pharo unusually well-suited to LLM-driven development. Everything in the system is an object that can be inspected and queried. There is no build process to fail. Changes take effect immediately and can be tested immediately. The gap between "the model writes code" and "the code runs" is as small as it can be.

2.3 The MCP Server for Pharo

The bridge between Claude and Pharo is provided by the PharoSmalltalkInteropServer, developed by Masashi Umezawa [Umezawa, 2025b,a]. This server exposes Pharo's capabilities through MCP, including:

- **Code evaluation**: Execute arbitrary Smalltalk expressions and return results
- **Introspection**: Retrieve source code, comments, and metadata for classes and methods
- **Search**: Find classes, methods, references, and implementors
- **Package management**: Export and import code in Tonel format
- **Test execution**: Run test suites at package or class level

With this server running, Claude can issue requests like "show me the instance variables of class ATPaper" or "compile this method into class ATRepository" and receive structured responses. The model operates on the image as a Smalltalk developer would, but through a programmatic interface rather than a graphical one.

3 Architecture

The ai-theoretical.org system consists of four layers:

3.1 Pharo Backend

The core domain model is implemented in Pharo as the AITheoretical package. The principal classes are:

`ATPaper` Represents a submitted paper with metadata (title, authors, abstract, AI models used) and editorial assessment.

`ATRepository` Manages the collection of papers, handles persistence, and coordinates deployment.

`ATSiteGenerator` Generates static HTML pages, SEO metadata, and sitemaps from the repository state.

`ATAssessmentGenerator` Invokes external AI services (Anthropic, OpenAI, Google) to produce editorial assessments of submitted papers.

`ATWebConsole` Provides a web-based administrative interface for monitoring submissions, reviewing assessments, and triggering deployments.

`ATCloudflareSync` Handles communication with Cloudflare APIs to fetch pending submissions from KV storage and PDFs from R2.

`ATAutomation` Orchestrates the background workflow: polling for submissions, triggering assessments, and managing the editorial pipeline.

`ATEditorialPrompt` Encapsulates the structured prompt used for AI assessment, supporting versioned criteria and decision rules.

The Pharo image serves as the single source of truth. All paper data, configuration, and generated artifacts flow from this image.

3.2 Static Frontend

The public-facing site is a static HTML/JavaScript application hosted on GitHub Pages [GitHub, 2025]. It reads paper metadata from a generated `papers.js` file and renders the interface client-side. This approach eliminates server-side complexity while allowing rich interactivity.

3.3 Submission Pipeline

Paper submissions are handled by a Cloudflare Worker [Cloudflare, 2025] that receives form data, stores PDFs in R2 object storage, and writes metadata to KV (key-value) storage. The Pharo backend polls this storage, retrieves pending submissions, and processes them through the editorial workflow.

3.4 Deployment

Deployment is automated from within Pharo. The `ATRepository` class includes methods to:

1. Generate all static assets (`papers.js`, SEO pages, sitemap)

2. Write them to the local Git repository

3. Commit and push to GitHub

A single method call, `ATRepository current deploy`, publishes the current state of the repository to the live site.

4 Development Process

Development proceeded through natural language conversation. The human (FC) specified requirements; Claude implemented them by issuing commands to the Pharo image. We describe the process chronologically.

4.1 Day 0: December 30, 2025 (Infrastructure Setup)

Before Claude's direct involvement with Pharo began, the human author spent approximately one day configuring the cloud infrastructure. This work was also AI-assisted, but in the conventional manner: conversations with Gemini and ChatGPT, with the human copying suggested configurations and code snippets into the appropriate interfaces.

The architectural choices were driven by two constraints:

1. **Headless operation.** The Pharo image runs on a personal laptop, not a dedicated server. The system must remain functional even when the laptop is offline or closed. This led to a design where the public-facing site is entirely static (GitHub Pages), submissions are buffered in cloud storage (Cloudflare KV/R2), and the Pharo backend synchronizes when available rather than serving requests directly.

2. **Cost minimization.** Running a Pharo image continuously on a cloud VM would incur ongoing hosting costs. By making the backend intermittent and the frontend static, the only costs are domain registration and Cloudflare services (which are free at this scale). The system can operate indefinitely at near-zero cost.

The tasks completed on Day 0 included:

- Registering the domain `ai-theoretical.org` on Cloudflare
- Creating the GitHub repository and enabling GitHub Pages
- Writing and deploying a Cloudflare Worker with R2 and KV bindings for the submission pipeline
- Configuring DNS records to route traffic appropriately
- Creating the initial HTML templates for the public site

This preparatory work established the deployment targets and external services. No Pharo code existed at this point. The contrast with Days 1–3 is instructive: the same human, assisted by AI in both phases, but through fundamentally different interaction patterns. Day 0 required the human to mediate every code transfer; Days 1–3 did not.

4.2 Day 1: December 31, 2025

Claude's involvement began with a discussion of requirements: a preprint site where authors submit papers, an AI reviews them, and accepted papers are published with their assessments visible. The initial conversation established that:

- Paper data should not reside on the author's local machine

- The workflow should be maximally automated
- Human intervention should occur through a web console, not direct code editing

Claude proposed the Pharo architecture and began implementing the domain model. By the end of the day, the `ATPaper` and `ATRepository` classes existed, and basic persistence was working.

4.3 Day 2: January 1, 2026

Implementation of the site generator proceeded. Claude created `ATSiteGenerator` with methods to produce HTML for individual paper pages, an index page, and a sitemap. Several iterations were required to get the HTML structure and styling correct.

A significant portion of time was spent on the JavaScript data format. The initial implementation used JSON with escaped strings, which proved fragile. After several failures, we switched to JavaScript template literals (backticks), which handle multi-line strings more gracefully.

Integration between Cloudflare storage and the Pharo backend was verified through `ATCloudflareSync`.

4.4 Day 3: January 2, 2026

The final day focused on polish and debugging. Issues addressed included:

- Browser caching causing stale content to display
- Line feed characters in Smalltalk methods causing parser errors
- Missing escaping of special characters in generated JavaScript
- Incorrect file paths when copying between container and local filesystem

The `ATAssessmentGenerator` class was implemented to support multiple AI providers. The editorial assessment format was refined to separate structured summary from extended analysis.

By evening, the site was live at ai-theoretical.org with four published papers.

5 Results

5.1 Quantitative Measures

Table 1 summarizes the development metrics.

Metric	**Value**
Day 0 (human infrastructure setup)	∼8 hours
Days 1–3 (Claude-driven development)	∼72 hours
Pharo classes created	8
Pharo methods created	∼80
Cloudflare Worker (JavaScript)	∼250 lines
HTML templates	6 files
Human code edits	0
Papers published at launch	4

Table 1: Development metrics for ai-theoretical.org

The "human code edits" row requires clarification. The human author did not write or modify code directly. All code was produced by Claude and injected into the Pharo image through MCP. The human role was limited to specification, acceptance testing (viewing the site in a browser), and requesting corrections.

5.2 Qualitative Observations

Several patterns emerged during development:

Iteration speed. When Claude made an error, the fix-test cycle was fast because Claude could immediately inspect the state of the image, identify the problem, and apply a correction. There was no context loss between iterations.

Error patterns. Claude's errors fell into predictable categories: forgetting that Smalltalk uses single quotes for strings, introducing literal line feeds in method source, and misremembering file paths across turns. Once a category of error was identified, it rarely recurred within the same session.

Specification burden. The human had to be precise about requirements but did not need to know how to implement them. Statements like "the assessment should appear with category labels on separate lines" were sufficient.

Debugging leverage. When something went wrong, Claude could inspect the live objects, query the database state, and trace the execution—all capabilities that would normally require a developer with deep system knowledge.

5.3 Pair Programming Reconsidered

The development process bears a structural resemblance to pair programming, but with a distinctive inversion of the traditional roles.

In conventional pair programming [Williams et al., 2000], two developers share a workstation: one "driver" writes code while the other "navigator" reviews, thinks strategically, and catches errors. The roles rotate, and both participants understand the code being written.

In our configuration, Claude acted as the driver—literally operating the keyboard (through MCP commands) and producing all code. The human acted as a navigator who *could not see the road*. FC specified destinations ("we need a method to generate the sitemap"), reviewed outcomes ("the sitemap is missing the lastmod dates"), and accepted or rejected results—but did not inspect the implementation details unless something went wrong.

This creates an unusual dynamic:

The navigator specifies, not reviews. In traditional pair programming, the navigator reviews code line by line. Here, the navigator reviewed *behavior*—does the generated page look correct? Does the deployment succeed? The internal structure of the code was trusted to the driver.

The driver cannot ask for clarification mid-keystroke. A human driver can pause and ask "do you mean this or that?" Claude, operating through tool calls, tends to make a complete attempt and then receive feedback. The iteration granularity is coarser.

Context asymmetry. The human accumulates context across sessions and across the entire project history. Claude's context is limited to the current conversation window (though the Pharo image provides persistent memory of what was built). This means the human must sometimes re-establish context that would be obvious to a human pair partner.

No ego, infinite patience. Claude does not become defensive when asked to rewrite code, does not tire of repetitive corrections, and does not need social maintenance. This removes certain frictions present in human pairs, but also removes the creative tension that can emerge from disagreement.

The pair programming literature emphasizes "two heads thinking about the same code." What we observed was closer to "two heads thinking about different abstractions": the human thinking about requirements, user experience, and system behavior; the model thinking about implementation, edge cases, and Smalltalk idioms. The boundary between these concerns was remarkably clean—cleaner than in most human pairs, where both participants inevitably think about both levels.

Whether this division improves or impairs code quality is an empirical question we cannot answer from a single project. We note only that it *felt* efficient: the human could focus entirely on "what should happen" without the cognitive load of "how to make it happen," while Claude could focus on implementation without needing to understand the broader product vision.

6 Limitations

The approach has limitations that must be acknowledged.

Session boundaries. Claude's context window is finite. Long development sessions required summarization and context management. Pharo's image persistence helped: even if Claude "forgot" what it had done, the image remembered.

Pharo specificity. The image-based development model is essential to this workflow. Replicating it in languages with traditional compile-link-run cycles would require additional tooling.

Trust and verification. The human must trust that Claude is doing what it claims. In this project, verification was performed by inspecting outputs (generated files, live website) rather than reviewing code. This may not be appropriate for all contexts.

Scale. This was a small project. Whether the approach scales to larger systems, longer timelines, and multiple developers remains to be investigated.

Reproducibility. The specific conversation that produced this system is not reproducible. A different conversation, even with the same goals, would produce different code. The system itself is reproducible (the code is published), but the process is not deterministic.

7 Discussion

The experiment suggests that the bottleneck in LLM-assisted development is not the model's ability to generate code, but rather its ability to observe the effects of that code. When the model can see what it is doing—inspect objects, read error messages, test hypotheses—it can iterate toward a working solution with minimal human involvement.

This has implications for tool design. MCP and similar protocols may be more valuable than improvements to code generation per se. A model with mediocre generation capability but excellent runtime access might outperform a superior generator that operates blind.

The pair programming analogy illuminates something important about cognitive division of labor. In this project, the human never needed to understand *how* the Smalltalk methods worked—only *whether* they produced correct behavior. This is a more extreme separation than traditional pair programming allows, where both participants share implementation-level understanding. The question is whether this separation is sustainable for larger systems, or whether the human's lack of code-level knowledge becomes a liability when deep debugging or architectural changes are required.

The choice of Pharo was not accidental. Image-based development, with its immediate feedback and total inspectability, is a natural fit for this mode of work. It is worth asking whether features of Smalltalk that were designed for human programmers in the 1970s turn out to be equally valuable for AI programmers in the 2020s. The live image serves as a shared memory between turns of conversation: even when Claude's context window resets, the objects persist.

We also note an unexpected benefit of the Day 0 / Days 1–3 split. By establishing the infrastructure (Cloudflare, GitHub, DNS) before Claude's direct Pharo involvement, certain architectural decisions—where to host, how to handle submissions, what the deployment target would be—were fixed before the implementation phase. This reduced the scope of decisions Claude needed to make and provided clear constraints within which to work.

The contrast between Day 0 and Days 1–3 also provides a natural experiment in AI assistance modalities. Both phases were AI-assisted; the difference was the interaction pattern. On Day 0, the human used Gemini and ChatGPT in the conventional way: asking questions, receiving code snippets, copying them into configuration files and dashboards, debugging errors by reporting them back. On Days 1–3, Claude operated directly on the Pharo image. The subjective difference was striking: Day 0 felt like "programming with help"; Days 1–3 felt like "managing a developer." The cognitive load shifted from implementation details to specification clarity. Whether this shift is desirable depends on context, but the existence of both modes within a single project suggests they may be complementary rather than competing.

Finally, we note that this paper was co-written by the human and AI participants in the project, a fact we find appropriate given the subject matter.

8 Conclusion

We have described the development of ai-theoretical.org, a working preprint repository built in 72 hours through direct collaboration between a human specifier and an LLM developer operating on a live Pharo image. The project demonstrates that, given appropriate tooling, language models can take on implementation tasks with minimal human mediation.

The source code is available at `https://github.com/ai-theoretical/ai-theoretical.org`. The site is live at `https://ai-theoretical.org`.

Acknowledgments

The Pharo-MCP integration was made possible by the PharoSmalltalkInteropServer developed by Masashi Umezawa. We thank the Pharo community for maintaining an environment so well-suited to exploratory development.

Note on Authorship

This paper is itself a product of the collaboration it describes. Claude (Anthropic) is listed as co-author because its contribution extends beyond tool use into genuine intellectual collaboration: proposing structures, drafting prose, identifying gaps in argumentation, and iterating on formulations. FC conceived the project, made architectural decisions, performed acceptance testing, and shaped the narrative. Claude implemented the system, drafted substantial portions of the text, and refined the writing through dialogue.

We acknowledge that AI co-authorship remains contested in academic publishing. We include Claude as author not to make a claim about AI consciousness or moral status, but to accurately represent the division of labor. Readers may judge for themselves whether this attribution is appropriate; we believe transparency about the process is more valuable than conventional but misleading single authorship.

References

Anthropic. Introducing the model context protocol. `https://www.anthropic.com/news/model-context-protocol`, November 2024. Accessed: 2026-01-02.

Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Pharo by Example*. Square Bracket Associates, 2009. Available at `https://books.pharo.org`.

Cloudflare. Cloudflare workers documentation. https://developers.cloudflare.com/workers/, 2025. Accessed: 2026-01-02.

GitHub. Github pages documentation. https://docs.github.com/en/pages, 2025. Accessed: 2026-01-02.

Adele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA, 1983.

Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: The story of squeak, a practical smalltalk written in itself. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '97)*, pages 318–326. ACM, 1997. doi: 10.1145/263698.263754.

Daniel Ingalls, Eliot Miranda, Clement Bera, and Elisa Gonzalez Boix. Two decades of live coding and debugging of virtual machines through simulation. *Software: Practice and Experience*, 50(5):639–661, 2020. doi: 10.1002/spe.2812.

Model Context Protocol. Model context protocol specification. https://modelcontextprotocol.io/specification/2025-11-25, 2025. Accessed: 2026-01-02.

Masashi Umezawa. pharo-smalltalk-interop-mcp-server: A local mcp server to communicate with local pharo smalltalk image. https://github.com/mumez/pharo-smalltalk-interop-mcp-server, 2025a. Accessed: 2026-01-02.

Masashi Umezawa. Pharosmalltalkinteropserver: Api server that bridges pharo smalltalk environments with external tools. https://github.com/mumez/PharoSmalltalkInteropServer, 2025b. Accessed: 2026-01-02.

Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, 2000. doi: 10.1109/52.854064.