Iterative Disambiguation: Towards LLM-Supported Programming and System Design

J.D. Zamfirescu-Pereira [1] **Bjoern Hartmann** [1]

Abstract

LLMs offer unprecedented capabilities for generating code and prose; creating systems that take advantage of these capabilities can be challenging. We propose an artifact-centered iterative disambiguation process for using LLMs to iteratively refine an LLM-based system of subcomponents, each of which is in turn defined and/or implemented by an LLM. A system implementing this process could expand the experience of end-user computing to include user-defined programs capable of nearly any computable activity; here, we propose one approach to explore iterative disambiguation for end-user system design.

1. Introduction

The aspiration to instruct computers in natural language has been a driving force for researchers for decades. Large language models (LLMs) provide a novel opportunity through their impressive capabilities for content manipulation and code generation and explanation. But while current LLMs like CHATGPT are proficient at iterating on content like text and code through natural language dialogue, end-users struggle to generate effective instructions (Zamfirescu-Pereira et al., 2023b), and the models themselves do not directly consider the content itself (text or code) as a "first-class" artifact. Lack of "first-class" status of the object of the work leads to a clunky user experience, as code or text content is spread across multiple chat messages instead of being injected directly into the document—or, if the proposed code or text directly replaces or is inserted into the artifact, the conversational context is lost.

These limitations restrict LLMs' usefulness for more complex activities, including working with longer documents and designing complex programs with many components. Approaches like chain-of-thought reasoning (Wei et al., 2023) and other LLM capability enhancements (see §4) can help LLMs develop and execute more complex tasks, but not in a way that generates a persistent, verifiable computational artifact, while LLM chaining techniques (Wu et al., 2022b) provide such an artifact, but require substantial technical knowledge to construct and debug.

2. Iterative Disambiguation

In this short paper, we note and suggest exploring an emerging artifact-centered *iterative disambiguation* approach to building systems in the LLM era, addressing the questions of whether and how we can use LLMs to transform a high-level user-specified goal into a component-based system architecture, where each component is in turn implemented or completed by an LLM, in a way that allows users to iterate on, validate, and execute that goal. Natural language instructions are inherently ambiguous—and LLMs can generate code, or prose, in response to ambiguous instructions only because they resolve that ambiguity with plausible, if generic, defaults. Through iterative disambiguation, users then identify and resolve those selected defaults that are inappropriate for the user's true goals.

When such a system is used for authoring *code*, assuming some user ability to evaluate the code driving the generated system is plausible—but for other domains, such as authors of arbitrary text, end-user understanding of the generated system's code and operation also seems critical to this process, and these users can quite reasonably wish to know: how do subcomponents work, and how are they interconnected?

For those writers, tools like SudoWrite, Notion, and Google's Docs Sidekick offer a menu of bespoke UIs for many different specific activities, often including an arbitrary option for "Custom" requests, which feed a prompt and selected context into an LLM. But these interactions are tightly scoped, and do not allow the abstraction and compositionality critical to complex systems—there is no saved library of prompt chains, and no way to describe processes that require multiple steps or additional interaction.

How deep is the "long tail" of desired activities, for writ-

[1]Computer Science Division, University of California, Berkeley, USA. Correspondence to: J.D. Zamfirescu-Pereira <zamfi@berkeley.edu>.

AI & HCI Workshop at the 40^{th} International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA. 2023.

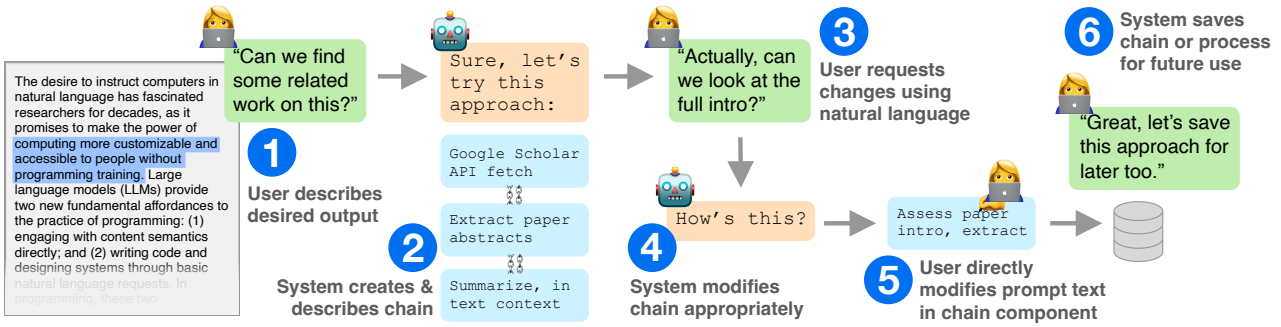

Figure 1. Example of one step of iterative disambiguation for a common research paper writing task, showing a user's initial request (1), the system's response (2), and the back-and-forth that supports iteratively refining and redefining the process encoded by the LLM in a "chain" (light blue boxes). Crucially, note that the user interaction is not limited to chat-style dialog (3), but can also directly manipulate prompts (5), code, or other aspects of the encoded process.

ers? Recent studies of users of generative AI assistance for writers (Gero et al., 2023) suggests there are many types of activities writers derive value from where an LLM could assist, but current tools limit this potential.

3. Example Uses

Consider the example shown in Figure 1. A user is seeking related work on a region of highlighted text; the system proposes one approach to finding such work, described in natural language, but backed by a mixture of ordinary code (Google Scholar API fetch) and LLM "chain"-type nodes (Extract paper abstracts; Summarize, in text context) that can be further inspected if desired. In response to this approach, the user recognizes that they in fact have something else in mind—the full intros, not just the abstracts—and communicates this request; the system responds in turn with a modification to one of the nodes. At this point, the user makes a direct manipulation to one of these prompts to better reflect her desired outcome.

Using traditional programming paradigms, writing such a program would require considerable effort, likely suppressing the quantity of similar programs that actually end up written. Unless used repeatedly, the time cost payoff period may be high enough to dissuade even fast programmers from pursuing such projects without a clear expectation of use in mind.

A second example appears in Figure 2; this pipeline describes a task that could be triggered daily, whose nodes echo common functional programming primitives, but which nonetheless can be constructed with the straightforward dialogue shown in Appendix A. In this pipeline, the user is requesting a list of all ArXiv submissions in HCI, extracting their titles and URLs, pulling their abstracts, and then filtering for relevance to LLMs.

4. Background & Related Work

Existing tools like PromptAid (Mishra et al., 2023) and PromptChainer (Wu et al., 2022a) allow technically-savvy end-users to build complex LLM-based systems without writing code, but users still must know how to break down their desired tasks and systems into the specific small subcomponents that LLMs enable, and then design prompts to complete those subcomponents—and these are challenging tasks; recent work suggests that prompt engineering itself is hard for both end-users (Zamfirescu-Pereira et al., 2023b) and experts (Zamfirescu-Pereira et al., 2023a).

The underlying concept of iteratively resolving errors with new data or instructions is far from new—and the "disambiguation" framing also underlies a major part of the programming-by-demonstration (PBD) workflow: users typically specify an initial set of examples or create an initial demonstration, and then iteratively "repair" the generated programs with new demonstrations or code changes, as in SUGILITE (Li et al., 2017), Ringer (Barman et al., 2016), and FlashFill (Gulwani, 2011).

Iterative refinement has also been put to good use in improving baseline LLM capabilities at writing code and QA tasks, in both fully-automated (Madaan et al., 2023; Kim et al., 2023; Gou et al., 2023) and human-in-the-loop (Chen et al., 2023; Akyürek et al., 2023; Paul et al., 2023) approaches.

These demonstrate that the basic principle can be used effectively, and further suggest that the major challenges are likely to be challenges of *interaction design* rather than model capability.

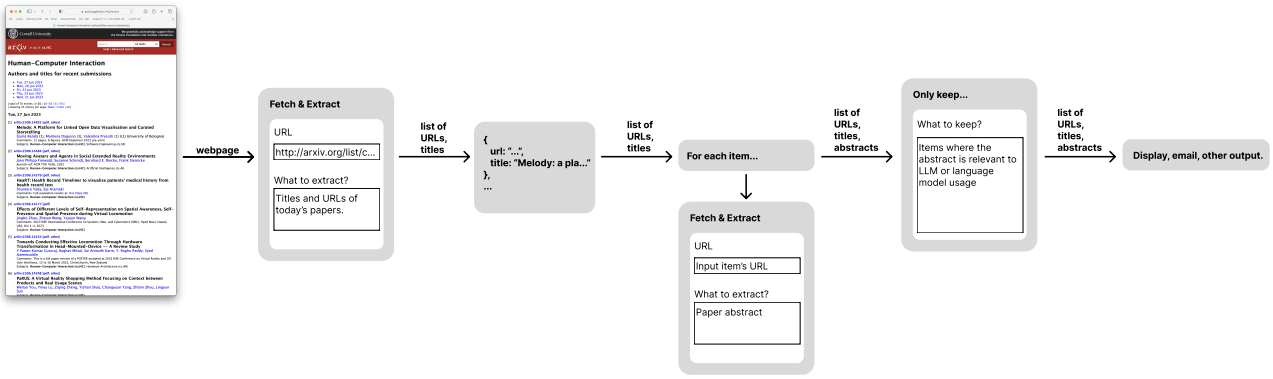

Figure 2. Example of a pipeline for data collection, generated by an LLM with a set of node names and descriptions as its initial prompt, combined with a user dialogue (see Appendix A).

5. Future Work: Implementing Iterative Disambiguation for Programs and Systems

We can use LLMs to help create chains of "components" (both LLM and traditional software) which, through careful use of abstraction and composition, can be designed, reasoned about, and extended by humans and LLMs, each with support from the other. A pilot implementation should focus on a specific target application and a specific kind of user; we suggest supporting authors in writing text. In domains like academic research writing, customized workflows could include integrating bibliographies, evaluating related work, streamlining and shortening paper sections, summarizing for particular audiences, and generating presentation outlines, among others.

Critically, these workflows should be authored through iterated natural language descriptions and dialog, while being instantiated by traditional software systems—and then described back to users across multiple modes: through natural language and visualizations, through examples of inputs and outputs, and through direct interaction with the designed system itself. The user and system together can then engage in iterative disambiguation of the expressed desired behavior. A user study could focus on users who have some level of technical sophistication (researchers, technical writers, of varying programming expertise), but not extensive experience with chaining LLMs, exploring users' naive expectations and approaches, evaluating the effectiveness of chosen abstractions and interfaces in enabling effective end-user system design and iteration, measuring goal achievement, and documenting users' behavioral patterns for future researchers.

Such a project would touch on a number of frontiers: end-user programming, novel user interactions, prototyping processes in design, and supplementing LLM capability with external tools. For example, developing ways to communicate code or code-like functionality (e.g., LLM chains) back to end-users and offering manipulation capability would inform new work on end-user programming; similarly, moving structured iteration beyond dialog and script-like periodic re-execution into artifact-centered direct manipulation by both human and AI would represent a substantial extension of mixed-initiative interaction (Horvitz, 1999). Finally, beyond research impact, expanding access to general-purpose computing has the potential to substantially alter *who* writes programs, and *why*.

References

Akyürek, A. F., Akyürek, E., Madaan, A., Kalyan, A., Clark, P., Wijaya, D., and Tandon, N. Rl4f: Generating natural language feedback with reinforcement learning for repairing model outputs, 2023.

Barman, S., Chasins, S., Bodik, R., and Gulwani, S. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications*, pp. 748–764, 2016.

Chen, A., Scheurer, J., Korbak, T., Campos, J. A., Chan, J. S., Bowman, S. R., Cho, K., and Perez, E. Improving code generation by training with natural language feedback, 2023.

Gero, K. I., Long, T., and Chilton, L. B. Social dynamics of ai support in creative writing. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–15, 2023.

Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan,

N., and Chen, W. Critic: Large language models can self-correct with tool-interactive critiquing, 2023.

Gulwani, S. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46 (1):317–330, 2011.

Horvitz, E. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 159–166, 1999.

Kim, G., Baldi, P., and McAleer, S. Language models can solve computer tasks, 2023.

Li, T. J.-J., Azaria, A., and Myers, B. A. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pp. 6038–6049, 2017.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback, 2023.

Mishra, A., Soni, U., Arunkumar, A., Huang, J., Kwon, B. C., and Bryan, C. Promptaid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models. *arXiv preprint arXiv:2304.01964*, 2023.

Paul, D., Ismayilzada, M., Peyrard, M., Borges, B., Bosselut, A., West, R., and Faltings, B. Refiner: Reasoning feedback on intermediate representations, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023.

Wu, T., Jiang, E., Donsbach, A., Gray, J., Molina, A., Terry, M., and Cai, C. J. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pp. 1–10, 2022a.

Wu, T., Terry, M., and Cai, C. J. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–22, 2022b.

Zamfirescu-Pereira, J., Wei, H., Xiao, A., Gu, K., Jung, G., Lee, M. G., Hartmann, B., and Yang, Q. Herding ai cats: Lessons from designing a chatbot by prompting gpt-3. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, DIS '23, pp. 2206–2220, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9781450398930. doi: 10.1145/3563657.3596138. URL https://doi.org/10.1145/3563657.3596138.

Zamfirescu-Pereira, J., Wong, R. Y., Hartmann, B., and Yang, Q. Why johnny can't prompt: how non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–21, 2023b.

A. Dialogue for Generating

In this section, we show the dialog that resulted in the pipeline shown in Figure 2, using GPT-4.

System Prompt You are a system design helper bot. You are helping a human build a system out of pretty standard parts. Here's a list of the parts that can be assembled into a workflow.

- `fetch` returns a text representation of a web page.
- `extract` pulls out a specific piece of information or a list of information from text, specified in natural language.
- `map` applies a transformation workflow (subworkflow, like a subroutine) to each element in an input list.
- `filter` applies a filter workflow to each element in an input list.
- `process` applies an arbitrary text transformation to a text input (`extract` is a special case of `process`), specified in natural language.

Users will describe their desired behavior, and it's up to you to figure out a workflow using the parts above. Sound good?

Assistant Absolutely! This sounds like a great challenge. Please go ahead and describe your desired system behavior, and I'll do my best to construct a suitable workflow using the components provided.

User I'd like to stay abreast of developments in HCI having to do with LLMs. The ArXiv preprint server is a great place to start. Can we create a workflow that looks at yesterday's posts and shows me the titles, authors, and abstracts of any papers having to do with LLMs?

Assistant Sure, I can help design a workflow for that using the components provided. Here's a proposed system design:

Fetch: Fetch the webpage representing yesterday's posts on the ArXiv preprint server.

Extract: From this webpage, extract a list of posts, where each post should include the title, authors, and the abstract.

Filter: Apply a filter to this list of posts to only include posts where the title or the abstract contains the keyword "LLMs". This filter could be defined by a process operation that checks for the presence of the term "LLMs" in the post content.

So, this would be the desired workflow:

```
fetch(arXiv_yesterdays_posts_URL)
extract("list of posts including title, authors, and abstract")
filter(process("post includes 'LLMs' in the title or abstract"))
```

Please note that this system assumes that the ArXiv preprint server makes its posts publicly available for fetching. You also may need to adjust the exact extraction and process operations to match the actual structure and content of the ArXiv web pages. Also, the details of the implementation will depend on the programming language and libraries you're using.

User Looks like the arxiv posts URL only includes titles and authors, not abstracts, but it does include a URL that can also be fetched for abstracts

Assistant Given this new information, we will need to adjust our workflow to include fetching the abstract from the individual paper's URL. Here's the adjusted workflow:

Fetch: Fetch the webpage representing yesterday's posts on the ArXiv preprint server.

Extract: From this webpage, extract a list of posts, where each post should include the title, authors, and the URL to the paper.

Map: For each post, fetch the webpage of the paper's URL and extract the abstract. This involves a nested workflow:

```
fetch(paper_URL)
```

```
extract("abstract of the paper")
```

Filter: Apply a filter to this list of posts to only include posts where the title, authors, or the abstract contains the keyword ”LLMs”. This filter could be defined by a process operation that checks for the presence of the term ”LLMs” in the post content.

So, the new workflow is:

```
fetch(arXiv_yesterdays_posts_URL)
```

```
extract("list of posts including title, authors, and paper URL")
```

```
map(fetch(paper_URL), extract("abstract of the paper"))
```

```
filter(process("post includes 'LLMs' in the title, authors, or abstract"))
```

This adjusted workflow should now handle fetching the abstracts for the papers and checking them for the keyword ”LLMs”.