AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts

Tongshuang Wu*
wtshuang@cs.washington.edu
University of Washington
USA

Michael Terry
michaelterry@google.com
Google Research
USA

Carrie J. Cai
cjcai@google.com
Google Research
USA

ABSTRACT

Although large language models (LLMs) have demonstrated impressive potential on simple tasks, their breadth of scope, lack of transparency, and insufficient controllability can make them less effective when assisting humans on more complex tasks. In response, we introduce the concept of *Chaining* LLM steps together, where the output of one step becomes the input for the next, thus aggregating the gains per step. We first define a set of *LLM primitive operations* useful for Chain construction, then present an interactive system where users can modify these Chains, along with their intermediate results, in a modular way. In a 20-person user study, we found that Chaining not only improved the quality of task outcomes, but also significantly enhanced system transparency, controllability, and sense of collaboration. Additionally, we saw that users developed new ways of interacting with LLMs through Chains: they leveraged sub-tasks to calibrate model expectations, compared and contrasted alternative strategies by observing parallel downstream effects, and debugged unexpected model outputs by "unit-testing" sub-components of a Chain. In two case studies, we further explore how LLM Chains may be used in future applications.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; *Interactive systems and tools*; • **Computing methodologies** → Machine learning.

KEYWORDS

Human-AI Interaction, Large Language Models, Natural Language Processing

---

*The work was done when the author was an intern at Google Inc.

1 INTRODUCTION

Large language models (LLMs) have introduced new possibilities for human-AI collaboration [10]. Pretrained on billions of inputs from the Internet [29], generative models like GPT-3 can now perform a wide variety of tasks [10], ranging from translation [12], to question answering [47], and even advanced story writing [60]. These successes are enabled by their ability to adapt to desired tasks purely using **prompts**, or natural language descriptions of the tasks [55]. For example, one could adapt an LLM to act as a translation engine, simply by providing a few examples of the desired inputs and outputs: "English: How are you? French: Comment allez-vous? English: Hello! French:" Based on this prompt, the model is likely to follow the pattern to output the correct French translation: "Bonjour!"

The relative ease of natural-language-based prompt programming suggests that LLMs may be useful assistants for real-world tasks, with users customizing the models to their own needs. In this light, recent work in Natural Language Processing (NLP) has begun to examine the algorithmic capabilities of LLMs, mostly on synthesized tasks [26, 55, 66]. However, many real-world tasks can be quite complex (*e.g.,* outlining long essays, debugging software code), and may present challenges for current LLMs to solve from a single model run. For example, as LLMs learn the forms of language [7], they produce lower quality outputs when solving tasks that require multi-step reasoning [11, 61, 67]. Likewise, they may fail to capture the subtleties of many tasks that involve multiple objectives simultaneously (*e.g.,* identifying *and* fixing multiple bugs in a code snippet). Figure 1 shows a task involving multiple concurrent objectives: (1) to rewrite peer feedback to be more friendly, and (2) to rewrite it with additional concrete suggestions, and (3) to ensure that each noted sub-problem (*e.g.,* too many words on slides, presentation meaders, does not engage with audience) is addressed. While an LLM can both generate suggestions [1] and adjust the tone in isolation (*e.g.,* in [3]), it lacks the capability to perform both tasks together well in an end-to-end manner. As a result, it produces a mediocre paragraph that only meets a few requirements (see output of Figure 1A).

Besides being inherently limited for complex problems, LLMs are also difficult to interact and collaborate with, as they can be opaque and hard to debug. Since LLMs can take in any natural language prompts, end users may struggle to determine how to change their prompts to remedy unexpected model outputs. They may also have difficulties developing accurate mental models of an LLM's capabilities and limitations. There are no obvious edits on the prompt that can, for instance, encourage the model to add more suggestions regarding "too much text on slides" in Figure 1A.

In this work, we introduce the notion of *Chaining* multiple LLM prompts together, to help users accomplish complex tasks with

Original feedback

> Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents.

A   NO-CHAINING

Direct rewriting

> *Given the following feedback, rewrite it into a friendly paragraph with concrete suggestions for each of Alex's presentation problems.*
>
> *Original feedback:*
> *[Original feedback text — see above]*
> *More friendly feedback:*
> [Model output — see below]

Split points   b1

Alex's problems
> Too much text...
> No clear structure
> Does not engage...

b2   Ideation

Suggestions for improvement
> More images on the slides
> Sectionalize the talk
> Ask the audience questions
> Use humor

B   CHAINING

Compose points   b3

Friendly paragraph

> Alex, you have too many words on your slides. You should use images and bullet points to help get your message across. You should have a clear structure for the presentation. You should also engage with your audience.

Friendly paragraph

> Alex, your presentation was interesting! However, I noticed that you have a lot of information on your slides. It might be helpful to vary pictures with text so that it is easier to follow. Also, you might consider the flow of your theme. If it were me, I would have divided it into three sections and then used your conclusion at the end. You may also want to add some humor, and ask more questions to engage the audience.
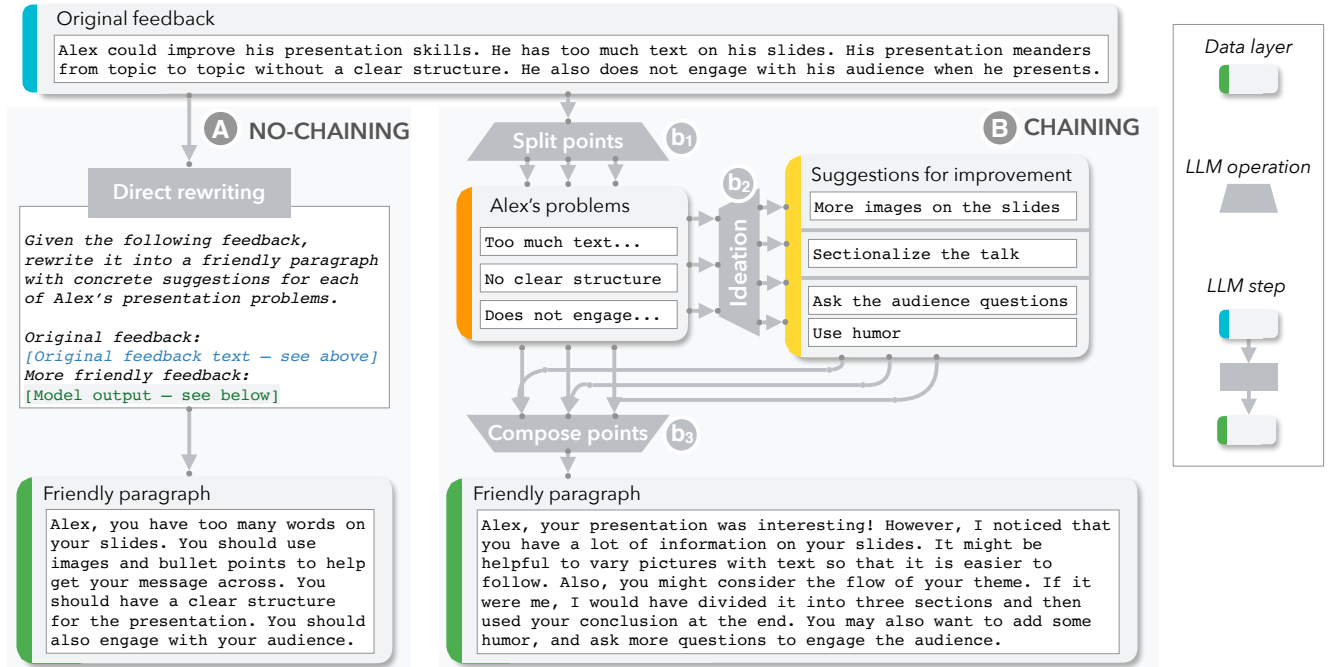
Data layer

LLM operation

LLM step

Figure 1: A walkthrough example illustrating the differences between *no-Chaining* (A) and *Chaining* (B), using the example task of writing a peer review to be more constructive. With a single call to the model in (A), even though the prompt (italicized) clearly describes the task, the •*generated paragraph* remains mostly impersonal and does not provide concrete suggestions for all 3 of Alex's presentation problems. In (B), we instead use an LLM Chain with three steps, each for a distinct sub-task: (b1) A Split points step that extracts each individual presentation •*problem* from the •*original feedback*, (b2) An Ideation step that brainstorms •*suggestions* per problem, and (b3) A Compose points step that synthesizes all the problems and suggestions into a final •*friendly paragraph*. The result is noticeably improved.

LLMs in a way that is more transparent and debuggable. Chaining takes advantage of LLMs' unique ability to handle a variety of independent tasks. In a Chain, a problem is broken down into a number of smaller sub-tasks, each mapped to a distinct step with a corresponding natural language prompt; results of one or more previous steps are aggregated in the next step's input prompt. Thus, Chaining enables users to run *the same* model on multiple sub-tasks, thereby granting each sub-task a higher likelihood of success (as opposed to solving the entire task in one go). In Figure 1B, while the underlying LLM remains the same, by *splitting* (*i.e.,*, extracting) presentation problems (b_1) and *ideating* suggestions per problem (b_2), the final *composed* paragraph (b_3) is more comprehensive in addressing all problems, and has a more constructive tone.

In addition to potentially improving outcomes, Chaining opens up new channels for fine-grained human feedback and control. For example, thanks to the separate *Ideation* step in Figure 1b_2, Chaining allows users to customize which suggestions to include in the final paragraph, an operation that is unavailable without Chaining (Figure 1A). We develop an interactive interface to expose these additional "knobs" to end users. The interface visualizes the Chain structure, and allows users to customize a Chain at various levels: they can iterate on the local prompts in each step, edit intermediate data between steps, or modify the entire Chain. To inform the design of this tool, we surveyed 73 existing LLM use cases and summarized them into a set of *LLM primitive operations*, each with

default prompting and data structures. They help inform what types of sub-tasks could be used within a Chain, as well as how those steps can feed into each other.

To evaluate the impact of Chaining on both task performance and user experience, we conducted a within-subject user study, in which 20 participants completed tasks using both Chaining and a standard (non-Chaining) interface, with the same underlying LLM powering all the steps in the Chaining interface, as well as the non-Chaining one. Our results show that Chaining significantly improved key dimensions of the human-AI experience: transparency, controllability, collaboration, and mental support. In addition, participants also achieved higher-quality outcomes ~82% of the time using Chaining. We also saw participants leveraging Chaining for purposes beyond immediate task accomplishment — they calibrated their expectations of the model using the smaller scope of sub-tasks, explored alternative prompting strategies by comparing parallel downstream effects, and debugged unexpected model output by isolating and "unit-testing" different parts of a Chain. Critically, these improvements were achieved *without changing the model itself*. These findings suggest that one way to improve the explainability and debuggability of an otherwise opaque, black-box LLM is to have it *do less*: breaking a problem up into smaller problems, having the model solve each (smaller) problem separately, showing the intermediate results, and allowing users to edit those results.

The ability to chain LLM calls using a set of Chaining building blocks, within an interactive interface, collectively represents a novel method and system for prototyping new AI-powered tasks and features using LLMs. We conclude the paper with case studies illustrating how Chaining can support more diverse applications in the future, as well as insights into challenges and opportunities that arose from our experiments. In summary, we contribute:

- We introduce the notion of *LLM Chaining*. Through a series of chained model calls, each targeting a small and well-scoped sub-task, we adapt *a single LLM* to contribute to *multiple sub-components* of a task.
- We design and implement building blocks for constructing and interacting with LLM Chains. These include a set of primitive LLM operations representing functions well-scoped for a single model run, and an interactive interface that displays the intra- and inter-step structures of a Chain. Users can run Chains step-by-step, and customize them at various granularities (editing intermediate model outputs, rewiring steps, etc.).
- We report results from a 20-person evaluation that shows Chaining can increase system transparency, controllability, and task outcomes. Importantly, these gains are achieved without any changes to the underlying model. Combined with the case studies, we demonstrate the potential of improving explainability and debuggability of LLMs through task decomposition and finer-grained application of LLM models.

Taken together, our findings inform the design and research of future human-LLM collaborative systems, an area of critical importance in years to come.

2 BACKGROUND AND RELATED WORK

2.1 Large Language Models

A generative language model is primarily designed to continue its input with plausible output (*e.g.,* given a prompt "I went to the", it might auto-complete with "coffee shop"). However, when pre-trained on billions of samples from the Internet, recent transformer-based LLMs [64] like GPT-3 [12] and Jurassic-1 [40] encode enough information to support additional *in-context learning*: they can be easily customized at run time (without any re-training needed) to handle new tasks beyond text continuation. To invoke the desired functionality, users write natural language instructions, or *prompts* [9, 43, 45], that are appropriate for the task. The most common patterns for prompting are either zero-shot or few-shot prompts. Zero-shot prompts directly describe what ought to happen in a task. For example, we can enact English-to-French translation with a prompt such as "Translate the sentence "Do you like the weather?" to French:". In contrast, few-shot prompts show the LLM what pattern to follow by feeding it examples of desired inputs and outputs: "[English] Hello! [French] Bonjour! [English] Do you like the weather? [French]". Given either of these prompts, the LLM may respond with the French translation "Vous aimez le temps?" [33]. Importantly, such task customization happens on the fly and, as a result, a single LLM can be flexibly adapted to a wide variety of use cases like code generation, question answering, creative writing, etc. [12, 60]. This flexible adaptation, together with the text-in, text-out structure, creates an intuitive natural language interface between humans and the model.

Despite their versatility, LLMs require careful prompt design. Various studies therefore focus on *prompt engineering* [9, 43, 45]. As manual prompting can be sub-optimal, some work automatically mines more effective prompts. However, the mined prompts tend to be less human-readable [58] and therefore less compatible with human-AI interaction. Conversely, strategies like progressive generation (*i.e.,* multi-round text expansion) [61] and meta-prompting (*i.e.,* asking the model to elaborate on the problem) [9, 55] attempt to seed LLMs to generate more effective prompts before solving the task. In essence, these approaches also adopt the spirit of multi-step problem solving, but focus on expanding the context *without* human intervention. Our work defines Chaining more comprehensively, with primitive operations that illustrate LLM capabilities, LLM steps that can add or remove information along the Chain, and editable intermediate data points.

2.2 Human-AI Collaboration

Human-AI interaction has been explored in domains such as classification [6, 59], drawing [24, 52], translation [28], creative writing [23, 27], and design ideation [36]. Prior work has noted core challenges of the interaction, such as a lack of transparency, controllability, and user agency [5, 13, 31]. Through Chaining, we aim to address these user-centered concerns.

In a collaboration, AI can play various roles, such as casual creators that encourage exploration [24] or assistants that compensate for human weaknesses [39, 70]. For example, Gero *et al.* [27] showed that generators could serve as cognitive offloading tools so that humans could focus their attention where it is needed most, a core motivation that we also share. Cai *et al.* [16] investigated how a medical AI can assist with doctors' decision-making process during prostate cancer diagnosis, by helping them compare and contrast similar images. Most of these studies, however, use task-specific models, and therefore limit observations to human interaction with AI that primarily serves one function, or in one domain (*e.g.,* writing, medicine, music, etc.). DuetDraw [52] may be an exception to this, as it uses several models, each of which supports a different co-drawing functionality. Rather than training multiple models for different tasks, or using a single model for a single type of task, our work explores how a *single* large language model (with inherently customizable capabilities) can support humans in *a variety of sub-tasks*. Finally, the closest work to ours might be online interfaces for users to interactively create prompts[1], or interfaces enabling users to perform natural language programming of code using a large language model [32]. These systems used prompt engineering to create a set of programming-related functionality for users. While this prior work focused on single prompts, our work looks at how Chaining *multiple* prompts can address a much wider range of human tasks, and evaluate its effects on user experience.

2.3 Workflows in Crowdsourcing

Though less prevalent in human-AI collaboration, the concept of Chaining is inspired by concepts of "pipelining" and "microtasking," which have long been used in crowdsourcing [15, 62]. In crowdsourcing, requesters break down complex tasks into pieces that can be performed independently, then combined [22, 34, 38, 54].

---

[1]https://gpt3demo.com/apps/openai-gpt-3-playground

Previous research shows that decomposed tasks allow the completion process to become more structured [21] and more resilient to interruptions [20], something we also witness in our user study. The goal of crowd workflows is typically to address and safeguard against the limitations of a typical worker. For example, Bernstein *et al.* [8] ensured text editing quality through a Find-Fix-Verify workflow, which modulates the scope of sub-tasks to reduce variance of crowdworker effort. Meanwhile, Context Trees [65] hierarchically summarize and trim the otherwise overwhelming global contexts, making them compact enough for a single worker to digest.

Our Chaining approach also aims to address pitfalls of a single LLM pass, but the pitfalls are somewhat distinct. While crowdsourcing focuses more on cognitive load and task duration — factors that can affect the performance of human workers [37] — for LLMs with intensive computing power, their limitations err towards a lack of reasoning abilities, high variance of prompt effectiveness, and exposure bias. A thorough analysis of these AI issues is needed for constructing and chaining LLM steps, which we illustrate in Section 3.1, and address through the design of primitive operations in Table 2. Through user studies (Section 5) and case studies (Section 6), we demonstrate that Chaining can effectively address these issues. Finally, our work also shares challenges found in crowdsourcing workflows, such as handling cascading errors that affect later stages [35], staged crash-and-rerun [42], all of which we take into consideration in the design of the Chaining structure. Beyond this, we advance the field by examining how core features of Chaining (*e.g.*, cascading effects, parallel paths) are used not only to accomplish tasks, but also to aid in increasing the transparency and debuggability of AI.

3 CHAINING LLMS

Despite the impressive capabilities of LLMs, there may be contexts in which LLM performance would suffer, such as if the data is formatted sub-optimally, if there is extraneous data in the input, if the task inherently demands solving multiple sub-parts, or if the user is asking the model to perform several tasks at once. Meanwhile, LLMs may perform *highly targeted* tasks well. By narrowing the scope and context of an LLM operation, for example, LLMs may themselves be useful for addressing some of their own challenges (*e.g.*, removing extraneous data, splitting problems into sub-parts, etc.). Thus, we hypothesize that decomposing a problem into smaller, highly targeted tasks is likely to increase model performance on those sub-tasks, and by extension, the overarching task.

We define *Chaining* as the process of breaking up complex tasks into smaller steps, where each step can be completed by an independent run of an LLM, and where the output of one or more steps is used as input for the next. To identify tasks that are most likely to benefit from Chaining, we first surveyed existing language modeling literature, and summarized **common challenges** LLMs face. As described in Section 3.1, these challenges are caused by the underlying modeling structure *shared* by the mainstream LLMs, including but not limited to GPT-3, Jurassic-1, and the internal LLM used in Section 5 and 6. Then, to identify promising sub-tasks that could be used as building blocks, we surveyed existing online demos of LLMs, and curated a list of **primitive LLM operations**, which

may help overcome those challenges by scoping the inputs/outputs to be more amenable to what an LLM can handle.

3.1 LLM Challenges & Primitive Operations

Existing literature exposes three main challenges that LLMs face:

- C.1 **LLMs lack multi-step reasoning capabilities**. Because LLMs are designed to grasp the *form* of language, rather than the *meaning* [7], they can struggle on tasks like sequential arithmetic problems, multi-hop question answering, recognizing and comparing sentences, or those that require branching logic [9, 11, 26, 66, 67].
- C.2 **LLMs suffer from *exposure bias*** [53, 61]. Because LLMs generate text sequentially in an autoregressive manner (the tokens generated by the models are themselves used to predict the next word), errors or imperfections from previous runs can accumulate. Thus, LLMs are less likely to perform well when generating long bodies of text. Exposure bias can also cause LLMs to produce redundant content, in some severe cases repeating the same phrase over and over again [30, 68]. As a result, they struggle to generate text with diverse themes or arguments (*e.g.*, suggestions for all three problems in the peer review example in Figure 1).
- C.3 **LLMs are sensitive to input prompts.** They tend to favor certain prompt formats, paraphrases [45, 51], or even certain information in the input. For example, prompts that are unnatural relative to the typical text distribution tend to be less efficient [11], while nouns and verbs are more important than adjectives and function words [51].

These challenges tend to stem from tasks being too broad. Yet, as discussed above, LLMs may be able to perform certain tasks well if they are highly targeted, with narrower contexts. Hence, with these challenges in mind, we reviewed 73 existing demos based on an extensive search of official LLM websites, social media, and published case studies (these are enumerated in Table 2, Appendix A) to identify promising LLM capabilities that may help scope the inputs/outputs, culminating in a set of primitive operations. Note that the operations we identified may not be exhaustive, but rather represent an interesting range for study, with a variety of operations addressing each LLM challenge. Pilot studies — as well as use cases we present later — suggested these were a reasonable set to pursue. Full details of our methodology can be found in Appendix A.

Table 1 shows how the derived operations fall into three categories and can address the aforementioned challenges. First, as LLMs may have difficulty applying common sense reasoning or complex inference to nuanced problems (C.1), the *Classification* operation can act as a **validation check** or triage, before more steps are carried out (Table 1a). For example, a chatbot may need to first classify the type of question a user is asking before providing adequate responses. Second, to alleviate exposure bias (C.2, the inability to generate long and diverse text), some operations can be used to **query small chunks of new content** (Table 1b), so as to gradually build up the generation diversity and length. Three ways to get new content include *querying facts*, *generating hallucinations*, and *ideating* lists of items. For example, in the peer review rewriting scenario (Figure 1B), the separate *Ideation* step per problem

(a) Validate and categorize the input

Def.	**Classification**: Assign the input to categories. Most useful for branching logic and validation.
| Ex. | ```
Classify if the question is answerable.
question: What is the square root of banana?
is answerable (Yes/No): No
``` |

(b) Gather additional information from the LLM

Def.	**Factual Query**: Ask the model for a fact.
| Ex. | ```
Given the US state, find the population.
US state: Washington
Population: 7.6 million
``` |
| Def. | **Generation**: Ask the model to do some creative "hallucination" on the input. |
| Ex. | ```
Given the topic, create a two-sentence horror story.
topic: Breakfast
two-sentence                    horror                    story:
He always stops crying when I pour the milk
on his cereal. I just have to remember not
to let him see his face on the carton.
``` |
| Def. | **Ideation**: Ask the model for a list of ideas or examples. |
| Ex. | ```
Given Alex's presentation problem, the following is a
list of suggestions.
Alex's problem: Too much text
Suggestions for improvements:
1) Use more graphics
2) Use bullet points
``` |

(c) Re-organize the input

Def.	**Info. Extraction**: Extract information from the context.
| Ex. | ```
Given the text, extract airport codes per city.
text: I want to fly from Los Angeles to Miami.
airport codes: LAX, MIA
``` |
| Def. | **Rewriting**: 1-1 mapping that changes the input to more machine-readable formats (*e.g.,* JSON to natural language). |
| Ex. | ```
Rewrite the first-person text into third-person.
first-person text: I decided to make a movie
third-person text: He decided to make a movie.
``` |
| Def. | **Split Points**: 1-N mapping that is particularly useful for splitting contexts. |
| Ex. | ```
Split the feedback paragraph into a list of Alex's
presentation problems.
Feedback: Alex could improve his presentation skills. He has
too much text on his slides. His presentation meanders from
topic to topic without a clear structure. He also does not
engage with his audience when he presents.
Alex's problems: 1) Too much text
2) No clear structure
3) Does not engage with audience
``` |
| Def. | **Compose Points**: N-1 mapping, the reverse operation of decomposition; merge multiple results back together. |
| Ex. | ```
Write one friendly paragraph to cover all the problems and
suggestions for improvement.
Alex's problems: 1) Too much text; 2) No...
Suggestions: 1) More images on the slides;...
Review: Your presentation was interesting!
However, I noticed that you have a lot of...
``` |

Table 1: We curate eight primitive operations that may be adequately handled by a single LLM run. Grouped according to their intended objectives, these operations can help address the LLM challenges detailed in Section 3.1. Along with the definitions, we provide examples of prompts that enact these operations, with the underlined text being the LLM output given the preceding prompt. The examples for *Ideation*, *Split* and *Compose* points are replicas of steps in Figure 1. The full implementations (with the parameters in Figure 6) are in Appendix D.

prevents suggestions for one criticism from being influenced by the other two criticisms. Finally, because LLMs may struggle with certain input prompt types, **reorganizing the prompt** could be helpful when its original form is convoluted. *Rewriting* and *Compose points* transform input into more parsable forms, *Information Extraction* elicits concise information (C.3), and *Split points* splits text into smaller and more manageable units (C.1)—all are summarized in Table 1c. As we will see in a case study (Section 6.1), translating JSON-formatted specifications to natural language descriptions helps LLMs parse the embedded information.

Chaining and its operations also have some parallels to crowdsourcing workflows. However, whereas sub-tasks in crowdsourcing are assumed to be feasible for a human worker (reviewed in Section 2.3), LLMs are more restricted in terms of tasks they can perform reliably, and thus the primitive operations presented are more scoped and granular. For example, Kittur *et al.* [35]'s Partition-Map-Reduce workflow uses *Split* and *Compose Points* operations (in Figure 1B), but does not indicate specifically how to transform the text (*Ideation*), though it also targets collaborative writing.

3.2   Designing Operations for LLM Chaining

An LLM Chain consists of multiple **steps**. Each step is defined by an **LLM operation**, which takes in input data and produces output data (which we call **data layers**). For example, the *Split point* operation in Figure 1 takes in the •*initial feedback for Alex* as input, and produces a list of •*presentation problems* ("too much text", "no clear structure", etc.) as output. LLM Chains are constructed by connecting these steps through shared data layers. In the same example above, the *Ideation* operation comes after the *Split points* operation, taking a (previously generated) •*problem* as input and producing •*suggestions for improvements* as output.

Each step of an LLM (an operation and its data layers) is accomplished through a natural language **prompt**. While prompts are task-dependent, they can have some task-agnostic properties. For example, the prompt for the *Classification* operation would likely contain the verb "classify", regardless of what is being classified. These keywords help set an LLM operation's scope and expectations [51]. We aim to abstract these task-agnostic properties into default parameters for each operation (Figure 2A), so as to provide consistent starting points for interacting with LLM Chains across use cases. Using the *Ideation* operation as an example, we show
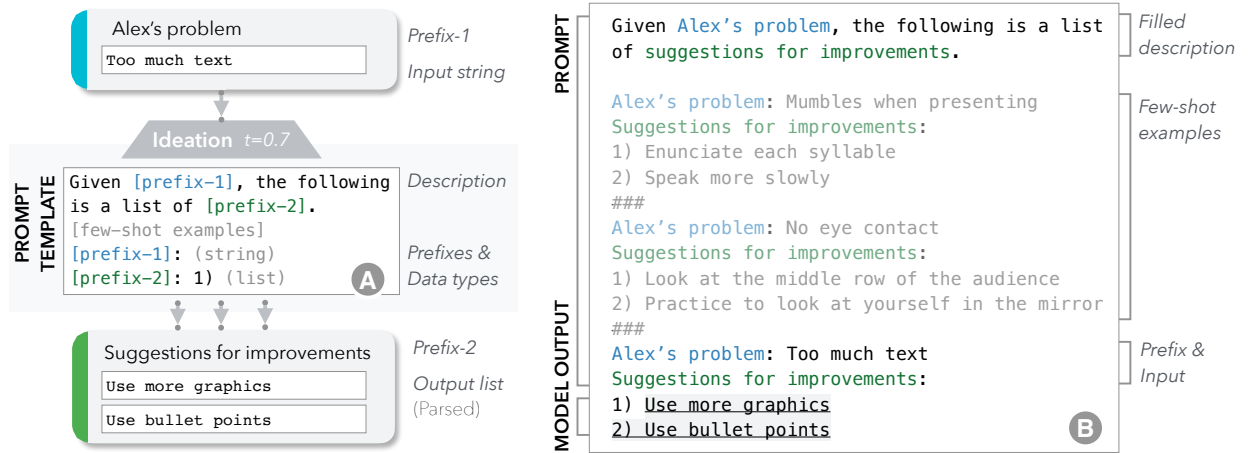
Figure 2: An example of how to create an LLM step using a prompt template (A), using the *Ideation* step of the peer review writing scenario (from Figure 1) as an example. For the peer review scenario, the *Ideation* operation takes in a problem (*e.g.,* too much text) as input, and produces suggestions for improvement as output, but the prompt template allows the *Ideation* operation to take in any custom inputs and outputs. The template includes placeholders for the input (prefix-1), output (prefix-2), and (optional) few-shot examples. (B) shows the actual prompt after filling in the placeholders in the prompt template.

how we design these parameters to satisfy the following three requirements for chaining, and how they help to build the *Ideation* prompt shown in Table 1 and Figure 2B.

Operations need to invoke the desired functionalities, through prompt design. To date, the most common patterns for prompting are either *zero-shot* or *few-shot* prompts, depending on how many demonstrating examples are provided in the prompt [12]. Zero-shot prompts directly describe what ought to happen in a task: *e.g.,* we can enact *Ideation* with a *task description* prompt "Given Alex's presentation problem, the following is a list of suggestions." In contrast, *few-shot* prompts show the LLM what pattern to follow by feeding it examples of the desired input and output data: "Problem: mumbles when presenting, Suggestion: enunciate each syllable, Problem: too much text, Suggestion:" (full prompt in Figure 2B). Given these prompts, the LLM might produce a reasonable suggestion, *e.g.,* "use more graphics on the slides." Zero-shot prompts can also be easily transformed into few-shot prompts, by appending examples to the initial zero-shot task description. In either case, prompts commonly include meaningful names as **prefixes** ("Problem:" and "Suggestion:") to demarcate structure, which helps re-emphasize the desired intent [64]. Following this convention, we build our prompts to include *task descriptions* followed by *prefixes*. Aside from the prompt itself, we also associate with each LLM operation a default **temperature setting**: a model parameter that influences the randomness of the LLM generation. For instance, creative operations like *Ideation* benefit from a higher temperature (t=0.7) than more factual or deterministic tasks like *Classification* (t=0.0) [2].

Operations should be able to take custom data layers as inputs and outputs. Though our walkthrough example takes in "Alex's presentation problem" and generates "Suggestions", in theory an operation should be able to handle any custom data layers. We thus create **prompt templates** to support a wide range of scenarios, with placeholders for input and output data. The template

allows us to build LLM steps simply by filling in the placeholders with definitions on data layers, as demonstrated in Figure 2. In particular, we include key verbs and nouns [51] in the template, to best reflect the operation objective (*e.g.,* "a list of" for *Ideation*, "classify" for *Classification*). The template also accepts optional few-shot examples. We can build the few-shot prompt in Figure 2B if we provide those pairs of problems and suggestions, or default to just the zero-shot version in Table 1 when examples are not readily available. Though we provide this as one example of a prompt template, we do not claim it to be exhaustive as there may be other equally effective ones.

Operations should handle parsing of the expected input/ output data types. Different data layers may take on different data types. For example, the Split step (Figure 1b_1) produces a *list* of problems, but only a *single* problem is the input to each subsequent Ideation step (b_2). To handle different formats in different steps, in each operation's definition, we define the required data types per operation (e.g. "list" in Figure 2 for *Ideation*), along with the corresponding parsing necessary to produce the expected data type (*e.g.,* split each row of the numbered list into an item).

Empirically, we find these defaults to work reasonably well across domains (see later sections 5 and 6). Still, we note that our defaults here are just one example of possible operation implementations; in our review of existing demos, there appeared to be many diverse prompting strategies even for the same task. We hope the prompt templates provided here may serve as a starting point for Chain designers or users to modify. In the next section, we demonstrate how these designs serve as the underlying data structure for interactive Chain execution by end-users.

4 INTERACTIVE USER INTERFACE

We designed an interface that helps users execute and customize LLM Chains interactively.
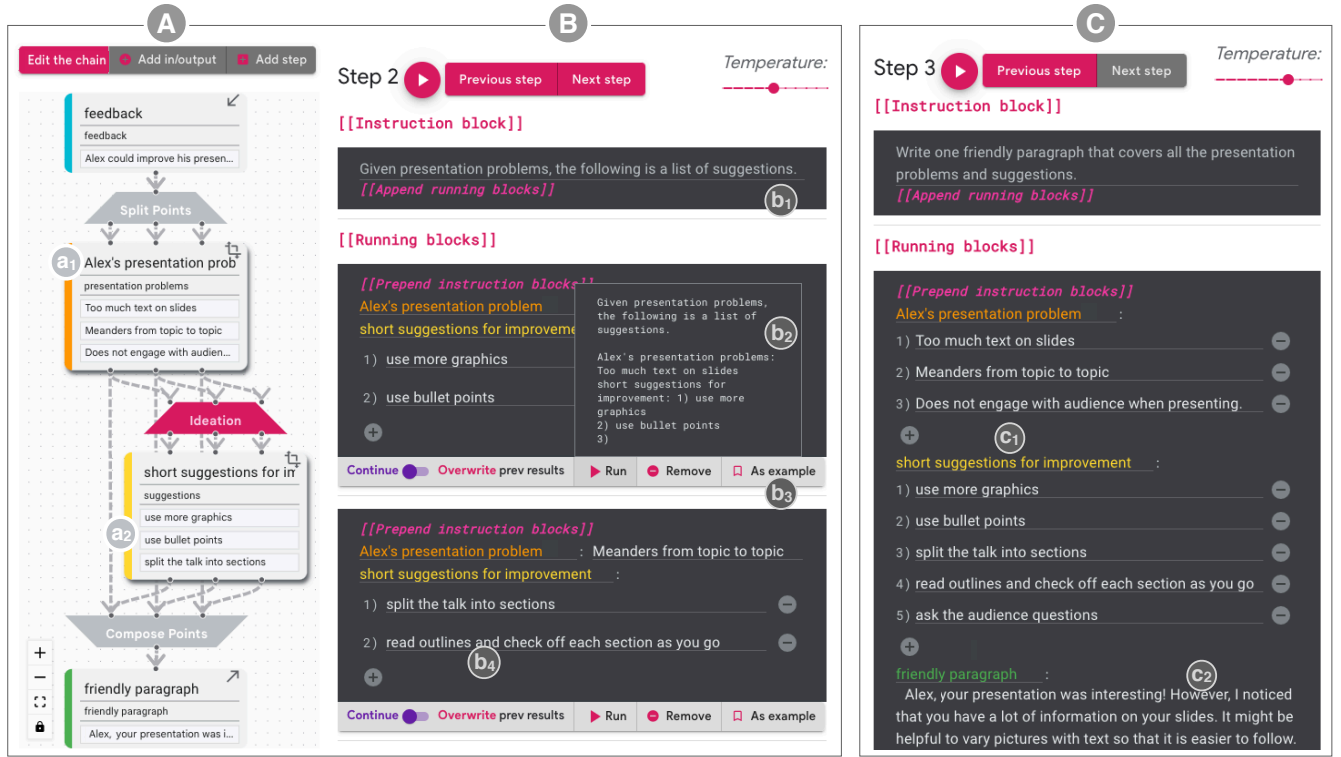
Figure 3: An overview of the interface, reflecting the peer review rewriting example in Figure 1. It consists of (A) a *Chain view* that depicts the high level Chaining structure, and (B/C) a *Step view* that allows for refining and executing each LLM step. The interface facilitates tracking the progress of the LLM Chain. For example, when moving from step 2: *Ideation* (B) to step 3: *Compose Points* (C), the previously generated presentation problems and suggestions become inputs for the final paragraph. A demonstration is available at https://youtu.be/QFS-1EWlvMM.

4.1 Design Rationales

Over the course of several weeks, we designed and iterated on the prototype with feedback from four pilot users (software engineers and designers who have experience designing LLM prompts), producing three design rationales for the final interface.

R.1 **Visually reflect the underlying Chaining structure.** In early prototypes, we explained the Chain structure using a static slide deck that highlighted the *data* produced at each step (*e.g.,* problems, suggestions for improvement, and final paragraph in Figure 1). In reaction, users expressed a desire to understand the *operations* taken at each step to arrive at these data layers (split points, ideation, compose points), and wanted to visually track progress through the Chain. To achieve this, we designed the interface to reflect not only the data layers, but also the LLM details within each step.

R.2 **Provide controls at different granularities.** Pilot users favored flexible controls. We observed users frequently making local fixes on intermediate data points that flow between LLM steps, and therefore designed the UI to allow in-place editing, without explicitly requiring a switch to editing mode. Some users also voiced an interest in iterating on alternative Chaining structures (*"Can I change this step with..."*). We therefore conclude that the interface should support modification of LLM Chains both locally (*e.g.,* changing one

task description or intermediate model output) and globally (*e.g.,* changing how the steps are connected). Because global changes have more impactful consequences (they may overwrite the underlying Chain structure), we designed the UI to require a switch to editing mode for this type of changes.

R.3 **The structured controls should still reflect the natural language interaction supported by LLMs.** In an early prototype, we formatted the data as structured tables with each data layer being a column, but received feedback from two users that making text edits in cells felt unnatural as they lost the sense of interacting with the model through natural language. To retain a natural interaction experience, we keep these structures as in-line text fields.

4.2 Interface Design and Implementation

We design the interface in Figure 3 following these design rationales above, which consists of two primary views: the *Chain view* (Figure 3A), and the *Step view* (Figure 3B/C).

The **Chain view** (Figure 3A) depicts the high level Chaining structure through a flow chart. It contains three primary visual cues that closely reflect the underlying design (R.1) described in Section 3.2. First, we use grey glyphs to represent *LLM operations*, with shapes indicating 1-1 (rectangle, for operations like *Rewriting* in Table 1), 1-N (trapezoid, *e.g., Ideation* operation), and N-1

data mappings (inverted trapezoid, *e.g., Compose points* operation). Clicking on these glyphs allows users to choose which step to zoom into (highlighted in pink), and the Step view would change in response. Then, we use rectangles with colored stripes to represent *data layers.* Users can preview their data entries through white rows (*e.g.,* Figure 3a_1 and a_2), which are updated after each LLM execution, and thus track Chain execution progress. Finally, we link these elements with dotted-line arrows to highlight which data output serves as the input to which step, and use the number of arrows going out of an operation to re-emphasize the data mappings (*e.g.,* multiple •***problems*** coming out from *Split points*, which is approximated with three lines, and a single •***paragraph*** out of *Compose points*).

On the right, the **Step view** (Figure 3B) allows users to explore each LLM step by interacting with inputs, outputs, and the underlying prompt structure. It is divided into an *instruction block* and several *running blocks* to handle *parallel paths*. Each of these parallel paths translates to a different LLM invocation; they share some common parts in their prompt strings, while having other parts being distinct from each other. We use the running blocks to hold the unique parts, and the instruction block to hold the shared sub-string is pre-pended to *all* running blocks, such that they are combined to form the full prompt. For example, Figure 3b_2 is the final prompt for the step that generations suggestions for the problem "too much text." It starts with the content from the instruction block (b_1), and merges the text in the running block thereafter, ignoring the other parallel running blocks.

Every running block visually resembles a textarea with a number of editable text fields. It shows the prefix fields before colons (*e.g.,•**Short suggestions for improvement**, c_1*) in the same color as the data layer rectangles, which helps users distinguish between data layers. It also includes text fields (b_4, c_2) for the model output for that step. The number of text fields (*e.g.,* 1 vs. N) are consistent with the data types defined for the primitive operation for that step.

This view also handles the per-step execution. Users can click the small "run" button to execute each running block individually. Alternatively, users can use the Play button on the top to run all the parallel blocks at once and compare their results. To improve natural language interaction transparency (R.3), running a block also triggers a preview of the final prompt text (b_2). The output is then parsed and added to the corresponding field (b_4, c_2) for users to further iterate on.

Interactions and controls. Notably, there are three levels of control available with this interface (R.2), from local customization of prompts to global modification of the LLM Chain structure, each with clear cues on its impact. First, users can customize the prompt for a particular step, *e.g.,* by changing its task descriptions. Since the customization only applies to the current step, all other views remain unchanged. Second, users can customize the model output for that step by adding, deleting, or editing content (*e.g.,* editing "read outlines" to *emphasize main points* in b_4), or rename data layers (*e.g.,* rephrasing "Alex's presentation problems" as "Criticisms of Alex" in a_1). These changes impact both the current step in focus as well as other steps involving the shared data layers (*e.g., Compose Points* takes in both the "problems" and the "suggestion" layer), and thus they can be changed either in the colored rectangles in

the Chain view, or through text fields in the Step view. Finally, users can more aggressively modify the Chaining structure itself by adding, removing and rewiring operations or data layers in the Chain view through intuitive visual programming (R.3). The change would then cause the entire Chain to re-render, with the defaults (*e.g.,* temperature, instructions) refreshed.

5 USER STUDY

To understand how Chaining affects the user experience of accomplishing tasks with LLMs, we conducted a within-subject user study comparing Chaining with a state-of-the-art baseline interface, on two user tasks.

5.1 Study Design

Underlying LLM. All of our experiments (including our baseline interface introduced below) and *each step of the Chaining interface* rely on exactly the same underlying LLM: LaMDA [63][2], a 137 billion parameter, general-purpose language model. This model is roughly equivalent to the GPT-3 model in terms of size and capability: it is trained with more than 1.5T words of text data, in an auto-regressive manner using a decoder-only Transformer structure which is useful for text generation. It has comparable performances with GPT-3 on a variety of tasks, and behaves similarly in its ability to follow prompts. Note that we only use this model to represent the recent class of LLMs; essentially, the chaining interface is model agnostic, and is compatible with any LLM that has in-context learning capability.

Systems. We compared Chaining with Sandbox, an interface that looks aesthetically similar to the Chaining interface, but without the Chaining functionality. We based the Sandbox interaction on GPT-3 playground,[3] the standard online interface for LLMs. It presents a single textbox with a run button, which allows the user to enter the text prompt, run the model on that prompt, and then view the model result in the same textbox, with the ability to edit that result and then continue to iterate. Like the Chaining interface, the Sandbox also allows users to adjust the temperature setting through a knob.

Tasks. We conducted the study using two tasks: peer review writing, and personalized flashcard creation, as they reflect different types of challenges (as explained below), and are both commonly used in user-centered task scenarios [14, 17, 25]. In the peer review writing task ("***Review***," our walk-through scenario), the user is given a paragraph (the same as in Figure 1) outlining three different problems in an imaginary person's presentation style, and their task is to write a friendly paragraph with 1-3 suggestions for each problem. In flashcard creation ("***Flashcard***"), participants were asked to create at least ten English-French sentence pairs they could use while traveling in Paris, and to make them as diverse as possible while being personalized to their own travel goals.

Though both tasks are possible when using an LLM without any LLM Chains, they present different types of challenges which could potentially be improved through Chaining. The *Review* task implicitly involves multi-step reasoning (Challenge C.1 in Section 3): to create a thorough and constructive review, one needs to identify

---

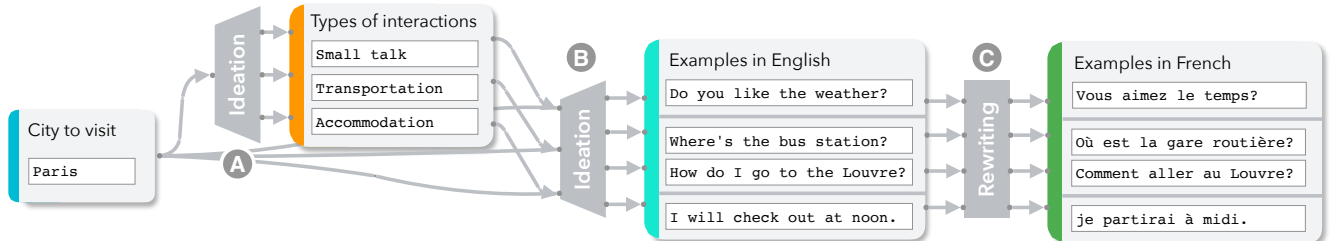[3]https://gpt3demo.com/apps/openai-gpt-3-playground

Figure 4: The LLM Chain for flashcard creation, with: (A) An Ideation step that brainstorms the •*types of interactions* that we might encounter when •*visiting a given city* (Paris), (B) Another Ideation step that creates a list of •*English examples* for each •*interaction type*, and (C) A Rewriting step that translates each •*English example* into •*French*.

each problem, provide suggestions per problem, and compose all the suggestions into one paragraph. The *Flashcard* task, on the other hand, exposes the challenge of having sufficient diversity in light of LLM exposure bias (C.2). In the Chaining condition, we built a default Chain for each task. The Chain for *Review* in Figure 1 reflects the three aforementioned steps (as explained before); the Chain for *Flashcard* (see Figure 4) sources additional content from the LLM like •*types of interactions* in a trip, which can help the user diversify the flashcards.

Study procedure. Before the study, participants completed a 30-minute tutorial that summarized the concept of LLMs and demonstrated how both Sandbox and Chaining work.[4] They were told upfront that both systems rely on the same underlying LLM. Then, in an hour-long study, participants performed a randomly selected task (*Flashcard* or *Review*), once with each interface (Sandbox and Chaining), whose orders were counterbalanced. We first briefed participants on the task, and then asked them to accomplish it with LLM's help in each interface until they were satisfied with the final results, or until they reached 25 minutes. Since LLM Chains came with automatically generated prompts (by filling in the templates), we similarly offered several default prompts for Sandbox that we knew to work reasonably, so that both interfaces had a fair starting point for prompt engineering (detailed in Appendix B). We encouraged participants to think aloud and describe their actions as they completed the task.

In the Chaining condition, participants were asked to first stick to the default Chain so that we could make consistent observations across participants in terms of how they *use* Chains. In the process, they could modify any other aspect (*e.g.,* the prompt, the intermediate model outputs, etc.) At the end, we gave participants the option to *modify* the default Chain, so that we could observe how they would expect the LLM to assist them beyond the default design. Finally, participants completed an exit survey and a semi-structured interview. They rated their experience using each interface along various dimensions. These dimensions were chosen to reflect the effectiveness of the human-AI collaboration (*e.g.,* support for their thought process, quality of the final result), and core user-centered challenges in human-AI systems [5, 13, 31] (*e.g.,* transparency, controllability, and sense of collaboration). They also verbally compared their impressions of the two interfaces, and envisioned possible use cases for them.

Collected data. We collected and analyzed three sets of data. First, to assess participants' **self-perceived experience**, we used a standard seven-point Likert Scale [41] to collect all ratings from the exit survey, with one being "Strongly disagree" and seven being "Strongly agree" with the statement in question (*e.g.,* for system Transparency: "The system is transparent about how it arrives at its final result"). Detailed survey questions are listed in Appendix B.1.

We also observed and recorded their entire task completion sessions, and later transcribed their comments and experience for qualitative analysis. Second, to quantify their **interaction mechanisms and behaviors**, we logged their interactions with the two interfaces. We were particularly interested in how participants reacted and iterated on model outputs, so we sorted their interactions with text fields by: (1) whether participants mainly relied on running the model again to get a different result (Consecutive run), or if they also edited the prompt in between (Edited); and (2) when they edited the prompt, how dependent it was on the existing model generation: whether they closely CURATED and refined the model outputs, loosely interacted around them by CREATING completely new content, or tried again by UNDOING the outputs. The detailed categorization criteria is in Appendix B.2. Third, to assess the **task outcome**, we logged the final reviews and flashcards participants created. Blinded to the condition, two non-participants performed anonymous, paired comparisons on results from each participant in Sandbox and Chaining, choosing the result that satisfied the task goals the best.

Participants. We recruited 20 participants using email lists that reach a wide range of practitioners (*e.g.,* UX designers, linguists, data analysts) at a large software company. Eight participants were 26-35 years old, eight aged 36-45, two aged 46-55, one 56-65, and one 18–26. As there is an initial learning curve associated with LLM capability, we required that participants had at least seen an LLM example before. Among those we recruited, half of the participants had no prompting experience but had seen online demos powered by LLM models, whereas the other half had some basic experience using default text prompts. Further, as the goal of Chaining is to use LLMs to assist with human tasks, we sought to recruit potential *users* of ML/LLM who would benefit from interacting with the models, rather than ML model experts or *creators*. Thus, our participants included technically knowledgeable but non-ML software engineers, linguists, UX designers, and data analysts who worked in a wide range of domains (*e.g.,* health, privacy, cloud storage, etc.).

---

[4]We took inspiration from the OpenAI: https://beta.openai.com/docs/introduction/prompt-design-101; the task used for tutorial is in Appendix B.3.

Each participant spent approximately 90 minutes total in our study, and received a $40 gift certificate for their time.

5.2 Quantitative Results: Increased Transparency & Control, and Higher-quality Task Outcome

All the participants were able to complete the tasks in both systems within the given time: they spent 12.4 ± 4.0 minutes in Sandbox, and 14.6 ± 5.4 in Chaining. Student's t-test did not show any significant difference between their completion time ($t = -.1.1, p = .278$). In analyzing subjective ratings from participants, the logged clickstreams, as well as the final generated results, we found:

First, Chaining *led to improved user experience in human-AI interactions.* We performed the non-parametric Wilcoxon signed-rank test to compare users' nominal Likert Scale ratings and, as shown in Figure 5, participants felt that Chaining helped them **think through the task better** (Chaining 6.0 ± 1.4 vs. Sandbox 3.6 ± 1.3, $z = 0, p < .001$), and **gave them more control** (6.2 ± 0.9 vs. 4.5 ± 1.3, $z = 3.0, p < .001$). They also rated Chaining as being more **collaborative** (5.7 ± 1.3 vs. = 4.6 ± 1.6 , $z = 25, p = .04$) and **transparent** (5.4 ± 1.3 vs. 3.8 ± 1.8, $z = 9.0, p = .002$).

Second, Chaining *shifted the types of edits participants made while interacting with the LLM.* In Chaining, participants were more likely to make manual interventions, whereas in Sandbox, they often re-ran the model (without changing the prompt) — akin to "rolling the dice again" in an attempt to get better output. As shown in Figure 6A, this tendency to perform consecutive runs without altering anything from the previous run occurred 51% of the time on average in Sandbox and 36% in Chaining. Student's t-test shows the difference is significant: $t = 3.5, p = .001.$[5]

The manual edits made were also finer-grained in Chaining than in Sandbox (Figure 6B). In Sandbox, people largely focused on either completely UNDO output and rerunning the model (45% of the time on average), or manually CREATING their own content as input to the model (14%). They only CURATED or modified existing text 41% of the time. On the other hand, in Chaining people performed CURATION 77% of the time, only doing UNDO and CREATE 18% and 5% of the time, respectively. The shift to CURATION is significant, according to Student's t-test ($t = -6.75, p < .001$).

As a result, Chaining *led to higher-quality generations that met the task goal.* The two independent raters consistently preferred Chaining results 85% and 80% of the time, respectively. The results also matched participants' own judgements in Figure 5 (see *Match goal*) — they preferred their own final results from Chaining (6.0 ± 0.9) to the Sandbox results (5.0 ± 1.1, Wilcoxon signed-rank test, $z = 11.0, p=.002$).

Aside from *using* Chaining, many participants were also able to iterate on and customize the underlying Chaining structure. While five of them preferred the default Chains provided and didn't want to change them, the remaining 15 people were able to identify parts they found lacking and suggested at least one change. 11 of them successfully implemented and executed one of their own solutions.

---

[5]The clickstreams fall into the continuous range of 0%–100%, and follows a normal distribution according to a D'Agostino-Pearson Test (*e.g.,* $p = 0.58$ for the ratio of consecutive runs).

5.3 Qualitative results: Chaining as Guardrails and Operation Manuals

Through analyses of the transcribed think-aloud comments and semi-structured interviews, we further unpack the reasons behind the quantitative differences. Since we asked participants to explain their Likert Scale ratings, their interview responses naturally map to dimensions in Figure 5 like transparency, collaboration, etc. One author further sorted their think-aloud comments into the categories. Three researchers then conducted thematic analysis, examining relationships between categories and iteratively converging on a set of higher-level themes. In general, Chaining helped support human-LLM interaction by serving as (1) a guardrail that helped users stay on track towards the task goal (Section 5.3.2 and 5.3.5); and (2) an "operation manual" that implicitly explained how to use LLMs for less obvious objectives, and that provided channels for users to intervene (Section 5.3.1, 5.3.3 and 5.3.4). In the following sections, we present key themes on how Chaining improved the human-AI experience, as well as some additional challenges brought on by Chaining.

5.3.1 Chaining helped users more fully capitalize on the model's latent capabilities.

In Sandbox, participants tended to use the LLM for a single purpose, under-utilizing the model's full potential in supporting various kinds of tasks. Four out of ten people in the *Flashcard* task only used the model as a translator in Sandbox, even though they were provided with default prompts that demonstrated how to generate English sentences using the model. In the *Review* task, even though nearly everyone (nine out of ten) used a two-step process of generating suggestions prior to merging them into the full paragraph (see the two-step prompt template in Appendix B.5), three people only relied on the LLM to generate suggestions, and then manually merged them into the paragraph themselves, without LLM input.

There may be two reasons for these behaviors. First, Sandbox naturally affords single-operation interactions. Given this, it is not surprising that users would gravitate toward using the model only for a part of the task that seemed most likely to yield promising results given the *status-quo* applications of machine learning (*e.g.,* translation), overlooking others that may seem less likely to succeed (*e.g.,* merging text into a paragraph). Indeed, some participants were unaware of less obvious sub-tasks (P4: *"this is just a simple translation task"* in *Flashcard*). Second, the friction of juggling multiple sub-tasks in Sandbox deterred some users from doing so. Even participants who became aware of the Chaining structure (from getting the Chaining condition first in their study condition order) struggled to replicate it using a single prompt. For example, P2 attempted to tackle both sub-tasks (generating diverse English sentences, and translating to French) simultaneously with a single prompt instruction: "Given the previous English sentence, translate it to French. Generate further English sentences relevant to travel in Paris." However, because the instruction was too nuanced for the model to follow, they eventually resorted to manually creating their own English sentences.

Ultimately, this inability to fully utilize the model led to lower quality final results in Sandbox. For example, the flashcards had less topical diversity (P4: *"I had limited diversity myself"*) because the *Ideation* step in Figure 4A was rarely ever leveraged. As a byproduct
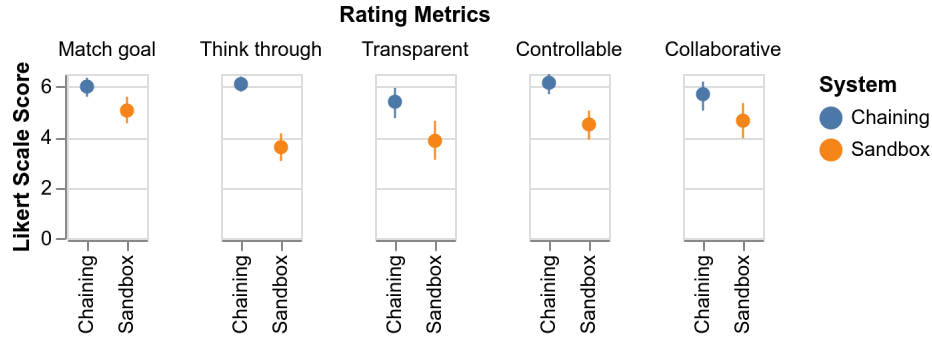
Figure 5: Participants' ratings in the form of seven-point Likert scale questions (details in Appendix B.1), with 95% confidence intervals. Using `Chaining`, participants felt they produced results that better *matched the task goals*, and that the system helped them *think through* the task. They also found `Chaining` more *transparent*, *controllable*, and *collaborative*.
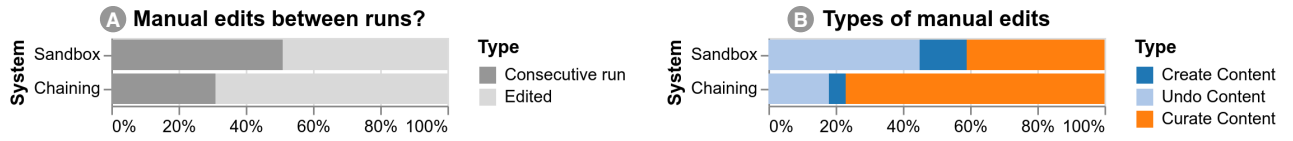

Figure 6: Distribution (based on the logged interactions) of how participants interacted with the prompts and model outputs, with and without chaining. (A) They made more *edits* in `Chaining` (compared to just repeatedly running the model), and (B) They tended to *curate* model outputs, rather than either deleting (*undoing*) them entirely or manually *creating* new content.

of the inadequate support, participants also found collaboration in Sandbox to be shallow (P5: *"I'm doing all the specific work [creating English sentences] and it's just doing its one thing [translation]"*). In contrast, `Chaining` allowed users to leverage the model in multiple ways. Seven participants particularly liked that they could accomplish multiple goals through the Chain, *i.e.,* acquiring model-powered diversity in the *Ideation* step, while maintaining translation correctness in the *Rewriting* step. This additional support may have contributed to participants shifting from creation (manually creating text from scratch) to curation (modifying model outputs) as shown in Quantitative Results (Figure 6B). Quoting P5, *"I didn't need to give it as much, but it was giving me a lot."*

LLMs' diverse primitive operations and capabilities also led participants to consider other ways the model might be helpful. For example, when asked to modify the Chaining structure itself, P1 in *Flashcard* swapped the *Ideation* step (which generated •*types of interactions*) with a *Generation* step to produce •*a journal of my one day trip*, so the model could *"think about what conversations can happen across my day trip"* and provide *"less generic context suggestions."* The operations became inspirational here. P12 and P20 in *Review* both added a *Classification* step to determine if the paragraph is in the right voice or if a suggestion is actionable, only once they realized the classification operation existed.

5.3.2 The ability to isolate interventions and save progress enhanced controllability of LLM.

Because each step of a Chain involves a separate run of the model, Chaining allowed users to control certain aspects of each sub-task *independent of* others. Four *Flashcard* participants in `Chaining`

noticed that the desired model randomness should vary per sub-task, and tuned the temperature settings accordingly: they increased the temperatures in *Ideation* steps to broaden the diversity and creativity of model responses (Figure 4A and B), and lowered it for *Rewriting* to increase the chances of getting correct model output (Figure 4C). However, none of them did so in the Sandbox condition (*e.g.,* P5: *"I realized my temperature was always high in sandbox. I should have had it low at translation, and high when I ask the model for English sentences."*) Many *Review* participants also liked iterating on each of the presentation problems individually (*e.g.,*"To much text on slides" vs. "No clear structure") without affecting the others.

This well-scoped impact of interventions may explain why participants felt more motivated and comfortable making manual edits in `Chaining` (Figure 6A). Nine people felt more compelled to enact controls on sub-tasks, knowing that they did not have to worry about unintended effects on other parts.

Four of them further noted that this clean separation would be tedious (if not impossible) in Sandbox, hence the differences in the perceived controllability in Figure 5. For example, P13 in *Review* attempted to replicate the exact same Chain in Sandbox. They manually divided the original paragraph into three problems, then asked the model for suggestions for each, and to compose the final paragraph. However, rather than storing suggestions externally and starting fresh for each problem, they simply stacked them together in a single prompt: "Original paragraph:...; Problem: too much text; Suggestions: 1)...; Problem: Split..." The resulting long and intertwined text became overwhelming: *"I was very nervous to edit anything, because I didn't know how that was going to impact the end task goals."*

Beyond staged interventions, staged outputs also provided participants with the opportunity to evaluate and improve individual

components irrespective of previous failure [50]. Three participants praised the ability to "freeze" their preferred intermediate data points: *"I reached some point of some progress in the middle of the Chain and if this works, then it's fixed when I play with the next step. It doesn't get lost — unlike the sandbox, where whenever I change something somewhere the result will be completely different"* (P10). Their observations are also in line with the crash-and-rerun capability of crowdsourcing [42], where local reruns are desirable without affecting previous stages.

5.3.3 Surfacing the Chaining structure increased transparency.
Chaining enriched system transparency, which helped participants better calibrate their expectations of the model. As each step of the Chain had a specific role (*Ideation*, *Rewriting*, etc.), they helped narrow the scope of the model's intended functionality, making it easier for participants to understand what to expect from a model that might otherwise seem all-encompassing. Nine participants noted this benefit of calibrated expectations. For example, P6 commented that *"Chaining helped you speak the language. It lift[ed] up the hood and showed you the steps and what's happening at different phrases,"* and P15 stated that *"having default settings like your templates gave me an idea of how it works."* As elaborated in Section 5.3.2, having isolated steps, each with a reduced scope, also enabled users to better anticipate the potential impact of their inputs, further increasing system transparency.

More globally, Chaining enabled users to develop a more accurate mental model of the LLM's capabilities, by allowing them to tinker with sub-components in a modular and comparative manner. Users could, for example, compare parallel paths to deduce how the model would respond to alternative inputs. In the *Flashcard* task, P8 noticed during the *Ideation* step that the model generated more useful English sentences when the •***types of interactions*** was "accommodation," compared to "topics related to public transportation." This hinted at the model's better performance when presented with a useful keyword. Modifying the order of LLM steps also enabled users to learn aspects of the model's strengths and weaknesses. When customizing the Chaining structure, five participants tried adding another *Rewriting* step either after the final paragraph (at the end of the Chain), or on the individual presentation problems (early in the Chain). Though initially unaware that LLMs can suffer from exposure bias (see C.2), participants quickly discovered through this comparison that the model could more effectively modify *sentences* than *paragraphs*. This comparison was rare in Sandbox, as it was not obvious to participants that they could keep the LLM functionality but shorten the input.

5.3.4 Surfacing the Chaining structure increased debuggability.
The increased transparency in Chaining also gave users better debugging mechanisms. When the model output was inconsistent with user intent, participants were at a loss for what to try next in Sandbox. Because users could conceivably type and modify any natural language prompt in the text box, the scope for "debugging" was too expansive. P9 remarked that *"too much freedom can be a curse,"* while P7 felt like *"sitting down in front of the controls of an airplane, all the knobs are there but I don't know what to do with them."* Instead, Chaining exposed intermediate knobs that helped participants draw a more direct connection between observed model

deficiencies, and possible remediation. P9 found it easier to debug by modifying the inputs and outputs for each step of the Chain, rather than merely re-running the model in Sandbox repeatedly, in the hopes of more promising model results (*"I had to constantly delete and rerun things."*). This may explain why the frequency of UNDO actions was reduced in Chaining (Figure 6B).

Accordingly, three interesting debugging mechanisms emerged: First, the isolated steps in Chaining acted as AI "unit tests" that enabled users to pinpoint a seemingly global error to its local cause. For example, participants in *Flashcard* frequently removed topics irrelevant to traveling (*e.g.,* education), so that sub-optimal solutions would not be fed into subsequent steps. Second, the ability to create parallel paths and alternate step orders (elaborated in Section 5.3.3) enabled *comparative debugging*. Revisiting the case mentioned above, observing a higher-quality path (*e.g.,* using a simple keyword in the prompt like "accommodation") helped participants infer how to improve prompts in other parts of the Chain (*e.g.,* changing "topics related to public transportation" to "public transportation.")

Finally, the ability to propagate a change throughout the entire Chain gave users *immediate feedback on whether a fix was successful*, thereby shortening feedback and iteration cycles. For example, P3 renamed •***types of interactions*** with •***places where conversation might occur***, so as to *"have flashcards grouped by happening at the airport, restaurant, while walking around streets."* They were impressed by the changes propagating to the final results: *"you can just change a step without affecting other steps but then your final results are reshaped based on that. I didn't think that was going to work that simply."* This combined ability to both isolate and propagate interventions was key to increasing AI debuggability.

5.3.5 Scoped objectives in sub-tasks served as guardrails against LLM-inspired tangents.
One challenge that hindered participants' performance on the tasks was LLMs' randomness and creative surprises. The model would often produce outputs that were compelling in their own right, which in turn would derail people from the intended task. For example, P5 in *Flashcard* was intrigued by an LLM-generated English sentence, "That man is suspicious to me," and started tricking the model into writing a story — *"I want to know what happened to the suspicious man!"* Five out of twenty people wandered from their task goal in Sandbox and began exploring tangents or attempting to "break" the model. They had to be reminded several times to get back on track. Participants later recalled their habit of drifting: *"I tried a lot of cool things, but it's not the task I want to complete"* (P17).

Interestingly, we found Chaining acted as a safeguard against model-inspired tangents, not only because each step of the Chain defined a clear goal, but also because the interconnected data layers motivated participants to deliberately steer outputs of each step away from cascading errors (*e.g.,* incorrect problem extraction in the first step of Figure 1b_1 could lead to a poor final paragraph). In the *Ideation* steps, participants would even manually move model output around to make sure they fit the topic (P7: *"this isn't really about asking for directions, I should put it in accommodation."*) Ultimately, participants treated the entire task more carefully (see Figure 5, *think through*) — *"if I was trying to do it with speed, I might*

find the sandbox easier; but if I want to do it with precision, I prefer the Chaining structure." (P13).

5.3.6 Additional challenges.

Chaining brought many benefits to human-AI collaboration, but it also presented several challenges. Nine participants noted that although they found the Chains to be transparent, rich, and educational, they were also more complex, with steeper learning curves. Moreover, while Chaining enabled participants to zoom into subtasks in modular ways, it also occasionally made the larger picture more difficult to recall: Four participants had questions about *"how my particular change to this data entry will affect the final result"* in Chaining (P2), and commented that the end-to-end aspect of Sandbox enabled them to see the direct effects of their actions. These challenges may have been a side-effect of participants using pre-defined Chains, which may not necessarily reflect their own intuition of how they would have decomposed the task [18, 71]. Most people had a much more fluent experience with the Chains they modified — *"I liked creating my framework."* (P13). Though beyond the scope of this paper, this raises the question of how to support users in not just using Chains, but also authoring their own Chains, to improve user agency and intuitiveness of Chaining [69].

Moreover, while Chaining provided better guardrails for staying on task, it may come at the expense of a decreased ability to explore freely; three participants mentioned they would prefer Sandbox for *"trying out random things and see if the model can cope"* (P3), and *"I feel more at liberty to play with language outside the the Chain"* (P6). They suggested they would prefer a combination of both systems: *"when there's more ambiguity I prefer the sandbox to explore first, but once I have a clear goal, I would use the Chaining to steer myself towards a fixed number of function blocks."* (P13)

Inspired by these concerns, we envision future research to focus on relaxing certain structural constraints and providing guidance on LLM Chain creation and refinement, which we detail later in Discussion (Section 7).

6 CASE STUDIES

Beyond the user study tasks, LLM Chaining has the potential to enable a wide range of complex applications. We illustrate how Chaining could support more diverse applications through two case studies in the domains of software development and accessibility, using the same model in our user study.

6.1 Case 1: Visualization code debugging

In this case study on visualization code debugging, we uncover how intermediate data points in a Chain can become useful, especially when the end goal of the task is unclear. Unlike typical code syntax errors, when a visualization violates design constraints [48], there are usually multiple valid solutions that cannot be objectively ranked. For example, the •*original visualization* (using VegaLite specifications [57]) in Figure 7 has a single violation, *i.e.,* circle size is continuous and thus should not be used to represent the discrete (nominal) field "Origin." However, there may be multiple ways to resolve the issue [19], such as using color instead of size (d_1), removing size information altogether (d_2), or changing the data encoded to a continuous "Acceleration" field (d_3). Thus, LLMs should reason about the violated constraints for users to adjust the

fixes. However, in a single run of an LLM, this reasoning can be challenging, as LLMs have trouble parsing visualization specs in JSON formats (see LLM Challenge C.3 in Section 3.1).

We thus created a Chain (see Figure 7) that (A) *rewrites* the JSON format in natural language, (B) *classifies* and validates the descriptions, and (C) *rewrites* the spec. To explore how the Chain performs in practice, we took examples from VizLinter [19], used five pairs of erroneous and fixed specs as few-shot prompt examples, and tested the Chain on another five cases. One author with sufficient visualization knowledge determined that the Chain correctly revealed the violated constraints for all the test cases, and provided useful fixes for two of them. We also tried running a single pass of the LLM for comparison on the same examples, using multiple prompt designs. We observed that output from the single-passes tended to be consistently worse, with at most one correct reasoning. This is possibly due to parsing difficulty (see LLM Challenge C.3), as well as the inability to disentangle the sub-tasks of validation and rewriting (C.1). In contrast, each Chain step was highly scoped, increasing the chance that the intermediate data would be correct.

6.2 Case 2: Assisted Text Entry

We further demonstrate how Chaining could enable the branching logic in *assisted text entry*. This is based on a real industry use case that aims to speed up gaze input by requiring fewer character inputs [4, 46, 56]. Ideally, a user (*e.g.,* person using Alternative and Augmentative Communication technology) would express a full sentence through short abbreviations that an LLM would automatically expand. However, there are too many possible expansions to disambiguate, *e.g.,*"LTSGCHKITOT" could mean "Let's go check it out ," "Let's get coffee and have a chat ," "Let's get some chicken in the old town," etc. Thus, the end user often needs to resolve the ambiguity or adjust the input.

With Chaining, we enable interactive disambiguation through gradual expansion and if-else logic. As shown in Figure 8, if the user input is a shorthand (*e.g.,*"LTSG"), the LLM should expand it to possible matching phrases ("Let's go", "Let's get"), which the user can select from. However, if the input is already a phrase, the LLM should instead auto-complete it ("Let's go" may trigger "check it out.") If the desired option does not appear, the user can also insert additional short-hands for the model to expand again, *e.g.,*"Let's go CHKITOT", which would exclude expansions starting with "Let's get." The switch between shorthand expansion and auto-completion enables better prediction on the full text, which would be nontrivial for a single prompt, given the different natures of the two branches. This case also provides a glimpse into how LLM Chains can help prototype applications with complex logic but simple interactions (elaborated in the next section).

7 DISCUSSION & FUTURE DIRECTIONS

Our work is a first step towards improving human-LLM interaction through Chaining. We found that it not only raises the ceiling of what LLMs can meaningfully support, but also boosts transparency, controllability and debuggability — key concerns when interacting with generative AI [5, 10]. Interestingly, we achieved this purely by reshaping the interaction mechanism, without any need to retrain the model. This suggests that LLMs to date may already have
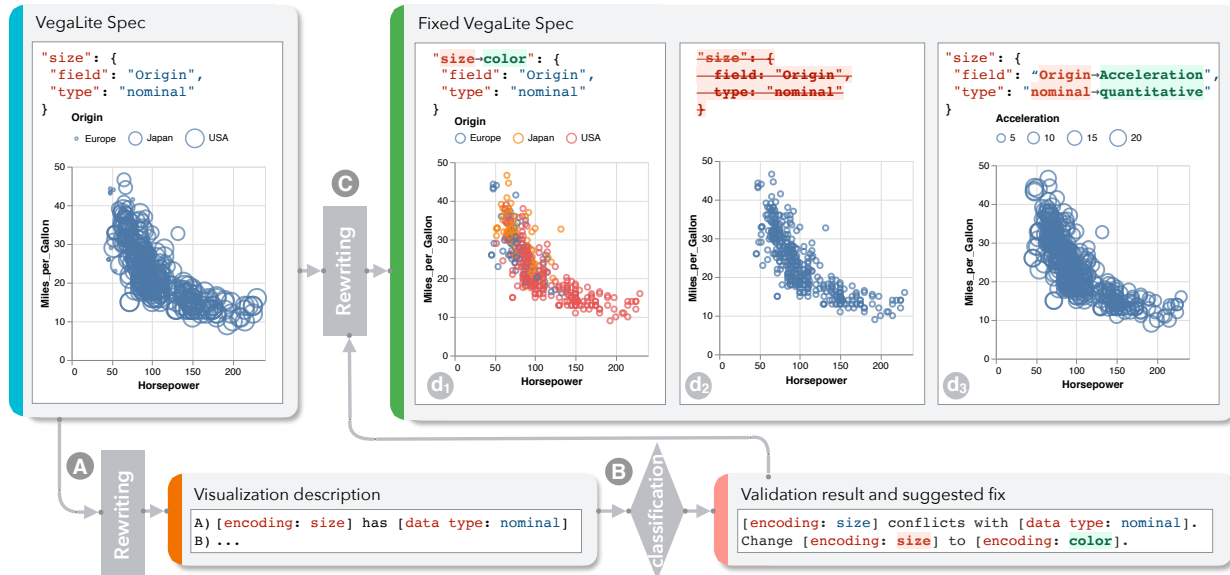
Figure 7: An example for Chaining-based VegaLite bug fixing (simplified; the full Chain is in Appendix C). (A) We first rewrite the •*JSON format specs* into •*natural language descriptions* to make it more parsable, then (B) classify the descriptions to •*validate design constraints and suggest fixes*, and (C) finally rewrite the •*final spec* based on the suggested fix. While the LLM generates the fix in d_1, users may also choose d_2 and d_3, both of which can fix the •*validated issue* just as effectively.
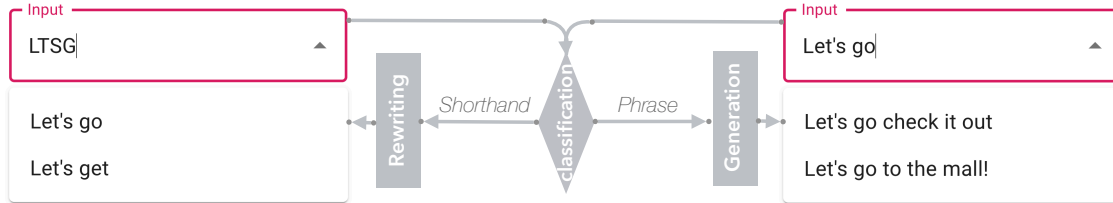

Figure 8: An example of Chaining-based assisted text entry (the full Chain is in Appendix C). To produce better full sentences, we classify the input text to switch between expanding shorthands (through Rewrite) and auto-completing phrases (through Generation). By wrapping the complex Chaining logic in a simple text field, we provide intuitive interactions for end users.

the potential to support human-AI collaborations on many complex tasks, if their latent potential can be better realized through thoughtful interaction design. Below, we discuss the implications of our studies, as well as future research directions.

Chaining as a new paradigm of control on multiple model units. Contrary to recent work in human-AI interaction, which primarily examined how to increase AI controllability through exposing knobs within a model [44, 49], our work opens up the possibility of steering AI using *the model itself* as units to control. In other words, beyond controlling properties within a single model unit, users may be able to achieve new kinds of control through manipulating how *multiple* model runs interact with one another, including: how modifications to upstream model units cascade, how to isolate changes between model units, and how to improve user inputs by comparing the effectiveness of parallel model runs. As language models grow in size and capability, they may ironically allow users to treat them as smaller entities of abstraction — serving as building blocks towards larger human goals.

We envision the HCI community innovating more types of building blocks that a model can provide, as well as the ways they can

be combined. In particular, model units could be used not only to accomplish sub-tasks, but also to more thoroughly aid in the task decomposition design and debugging process. To overcome users' own systematic omissions [70], an upstream unit could be designed to help users create sub-tasks to begin with, similar to metaprompting [55]. Or, model units could serve as checkpoints along the Chain to ensure data correctness (similar to assertions in code). Moreover, while the Chains in this paper consisted of only LLM steps, alternative designs may also interleave LLM steps with human-computation steps, depending on which roles each collaborator could best fill.

Chaining for rapid prototyping of integrated applications. Chaining also opens up new possibilities for designing AI-infused applications. With LLMs' easy adaptation to natural language prompts, users could conceivably already prototype custom ML functionality with lower effort, as they bypass the otherwise necessary but expensive process of collecting data and designing models upfront [10]. Chaining further accelerates this design process. Taking advantage of interactions between multiple LLM steps, developers could build multiple Chains to envision possible flows of how an application

may be used, and then perform A/B testing on those Chains. For example, in the case of assisted text entry (Section 6.2), developers could quickly prototype what might happen if end users were allowed to provide more context: *e.g.,* if the user is "having a meeting in 5 minutes," then "Let's go" is more likely than "Let's get" for the abbreviation "LTSG." They could test this interaction by adding an additional layer of input to the shorthand expansion step.

One might argue that, because each run of an LLM involves some computational overhead, chaining may introduce additional costs that need to be weighed against their benefits. However, as indicated above, a key benefit of chaining is that it could flexibly power a wide range of prototypes and applications, *without* the need to train or build bespoke, single-purpose AIs. Thus, we believe the saved efforts outweigh the cost.

Balancing between structured scaffolding and free exploration. While Chaining provided guardrails and scaffolding for helping users accomplish the task at hand, it also limited their ability to explore freely. Yet, experimenting, tinkering, and interacting are key to users forming mental models for AI [49]. One way to balance between structure and exploration is to loosen structural constraints *within* steps. For example, it may be useful to permit users to customize prompts within each step in a Sandbox-like environment, and to define their own input and output parsers. In other words, rather than providing a full implementation of steps, a Chain could define the API with input-output types, and ask users to fill in the implementations for each step. Or, a small Sandbox could be provided along-side the Chaining interface, for users to occasionally use when they need to experiment with a new approach.

Meanwhile, though our studies mostly explored how humans *use* pre-defined LLM Chains, a natural follow-up question becomes whether end users can effectively *author* their own LLM Chains. Indeed, one potential downside of Chaining is that it may *decrease* transparency if the pre-built Chain does not match the way a user would naturally break down the task (mentioned in Section 5.3.6). We believe our operations can serve as a starting point for future work on authoring. With the templates, users could instantiate an LLM step by defining the data layers and selecting the operations. In our study, most participants were able to spot deficiencies and refine the default Chains accordingly. Thus, we envision that a set of generic default Chains could help onboard end users to the idea of LLM Chaining, and inspire them to author more tailored Chains. We leave end user authoring of Chains to future work.

Enhancing LLM Chain design and refinement. Our work centered mostly on moderately complex tasks that can be naturally broken down. However, decomposition might be less straightforward in some cases [34]. Tasks with more complex interdependence may lose coherence and quality if they are split into independent subparts. For example, in the Review task (Figure 1), we treated the different problems independently. However, if the problems are interrelated, keeping them together would promote more effective suggestions (*e.g.,* not engaging *and* speaks too quietly). Moreover, while users had the option of excluding specific data layers along the way (*e.g.,* the original review in Figure 1 is not fed into the final step), the information loss may also lead to task distortion or compression [55]. In light of these issues, future work could investigate how to assist users in crafting the steps of a Chain

to maximize its utility [35]. For example, users could be provided strategic guidance on iterative Chain improvements, such as using paired comparisons and version control of Chain edits to help users decide whether to keep or further decompose an existing step.

8 CONCLUSION

In this work, we introduce the notion of "Chaining" multiple LLM steps together, such that the output of one step is the input to the next. We present an interactive system where users can modify these Chains, along with their intermediate results, in a modular way. We find that Chaining not only enhanced the quality of the task outcome, but also improved user satisfaction, with an increased sense of control and collaboration, a greater perception of transparency of the LLM system, and more support of the user's thought processes. Furthermore, we envision with case studies that LLM Chaining may be advantageous for complex AI-infusion applications and in cases where intermediate reasoning is more important than the final output. We encourage future work to explore how LLMs can serve other kinds of building blocks, how Chains can be used in rapid prototyping, and strategies that can help users build and iterate on Chains.

REFERENCES

[1] [n. d.]. GPT-3 is an idea machine. https://interconnected.org/home/2020/09/04/idea_machine. Accessed: 2021-08-23.
[2] [n. d.]. Prompt design 101. https://beta.openai.com/docs/introduction/prompt-design-101. Accessed: 2021-08-07.
[3] [n. d.]. A Recipe For Arbitrary Text Style Transfer With Large Language Models. https://www.gwern.net/GPT-3. Accessed: 2021-08-01.
[4] Jiban Adhikary, Jamie Berger, and Keith Vertanen. 2021. Accelerating Text Communication via Abbreviated Sentence Input. In *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conf erence on Natural Language Processing*.
[5] Saleema Amershi, Daniel S. Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi T. Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 3. https://doi.org/10.1145/3290605.3300233
[6] Gagan Bansal, Tongshuang Wu, Joyce Zhou, Raymond Fok, Besmira Nushi, Ece Kamar, Marco Tulio Ribeiro, and Daniel Weld. 2021. Does the whole exceed its parts? the effect of ai explanations on complementary team performance. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1−16.
[7] Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 5185−5198.
[8] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. 2010. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. 313−322.
[9] Gregor Betz, Kyle Richardson, and Christian Voigt. 2021. Thinking Aloud: Dynamic Context Generation Improves Zero-Shot Reasoning Performance of GPT-2. *arXiv preprint arXiv:2103.13033* (2021).
[10] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma

Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Kohd, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the Opportunities and Risks of Foundation Models. arXiv:2108.07258 [cs.LG]

[11] Gwern Branwen. 2020. GPT-3 creative fiction. (2020).

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[13] Daniel Buschek, Lukas Mecke, Florian Lehmann, and Hai Dang. 2021. Nine Potential Pitfalls when Designing Human-AI Co-Creative Systems. arXiv preprint arXiv:2104.00358 (2021).

[14] Carrie J Cai, Philip J Guo, James Glass, and Robert C Miller. 2014. Wait-learning: leveraging conversational dead time for second language education. In CHI'14 Extended Abstracts on Human Factors in Computing Systems. 2239–2244.

[15] Carrie J Cai, Shamsi T Iqbal, and Jaime Teevan. 2016. Chain reactions: The impact of order on microtask chains. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. 3143–3154.

[16] Carrie J Cai, Emily Reif, Narayan Hegde, Jason Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda Viegas, Greg S Corrado, Martin C Stumpe, et al. 2019. Human-centered tools for coping with imperfect algorithms during medical decision-making. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 1–14.

[17] Julia Cambre, Scott Klemmer, and Chinmay Kulkarni. 2018. Juxtapeer: Comparative peer review yields higher quality feedback and promotes deeper reflection. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. 1–13.

[18] John M Carroll and Judith Reitman Olson. 1988. Mental models in human-computer interaction. Handbook of human-computer interaction (1988), 45–65.

[19] Qing Chen, Fuling Sun, Xinyue Xu, Jiazhe Wang, and Nan Cao. 2021. VizLinter: A Linter and Fixer Framework for Data Visualization. IEEE transactions on visualization and computer graphics (2021).

[20] Justin Cheng, Jaime Teevan, Shamsi T. Iqbal, and Michael S. Bernstein. 2015. Break It Down: A Comparison of Macro- and Microtasks. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015, Bo Begole, Jinwoo Kim, Kori Inkpen, and Woontack Woo (Eds.). ACM, 4061–4064. https://doi.org/10.1145/2702123.2702146

[21] Lydia B Chilton, James A Landay, and Daniel S Weld. 2016. Humortools: A microtask workflow for writing news satire. El Paso, Texas: ACM (2016).

[22] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. 2013. Cascade: crowdsourcing taxonomy creation. In 2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker (Eds.). ACM, 1999–2008. https://doi.org/10.1145/2470654.2466265

[23] Elizabeth Clark, Anne Spencer Ross, Chenhao Tan, Yangfeng Ji, and Noah A Smith. 2018. Creative writing with a machine in the loop: Case studies on slogans and stories. In 23rd International Conference on Intelligent User Interfaces. 329–340.

[24] Nicholas Davis, Chih-PIn Hsiao, Kunwar Yashraj Singh, Lisa Li, and Brian Magerko. 2016. Empirically studying participatory sense-making in abstract drawing with a co-creative cognitive agent. In Proceedings of the 21st International Conference on Intelligent User Interfaces. 196–207.

[25] Darren Edge, Elly Searle, Kevin Chiu, Jing Zhao, and James A Landay. 2011. MicroMandarin: mobile language learning in context. In Proceedings of the SIGCHI conference on human factors in computing systems. 3169–3178.

[26] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. Minds and Machines 30, 4 (2020), 681–694.

[27] Katy Ilonka Gero and Lydia B. Chilton. 2019. Metaphoria: An Algorithmic Companion for Metaphor Creation. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 296. https://doi.org/10.1145/3290605.3300526

[28] Spence Green, Jason Chuang, Jeffrey Heer, and Christopher D Manning. 2014. Predictive translation memory: A mixed-initiative system for human language translation. In Proceedings of the 27th annual ACM symposium on User interface software and technology. 177–187.

[29] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300 (2020).

[30] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. https://openreview.net/forum?id=rygGQyrFvH

[31] Cheng-Zhi Anna Huang, Hendrik Vincent Koops, Ed Newton-Rex, Monica Dinculescu, and Carrie J Cai. 2020. AI song contest: Human-AI co-creation in songwriting. arXiv preprint arXiv:2010.05388 (2020).

[32] Ellen Jiang, Edwin Toh, Alejandra Molina, Aaron Donsbach, Carrie J Cai, and Michael Terry. 2021. Genline and genform: Two tools for interacting with generative language models in a code editor. In Adjunct Publication of the 34rd Annual ACM Symposium on User Interface Software and Technology.

[33] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. arXiv preprint arXiv:1602.02410 (2016).

[34] Joy Kim, Sarah Sterman, Allegra Argent Beal Cohen, and Michael S Bernstein. 2017. Mechanical novel: Crowdsourcing complex work through reflection and revision. In Proceedings of the 2017 acm conference on computer supported cooperative work and social computing. 233–245.

[35] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. 2011. Crowdforge: Crowdsourcing complex work. In Proceedings of the 24th annual ACM symposium on User interface software and technology. 43–52.

[36] Janin Koch, Andrés Lucero, Lena Hegemann, and Antti Oulasvirta. 2019. May AI?: Design Ideation with Cooperative Contextual Bandits. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 633. https://doi.org/10.1145/3290605.3300863

[37] Anand Pramod Kulkarni, Matthew Can, and Bjoern Hartmann. 2011. Turkomatic: automatic, recursive task and workflow design for mechanical turk. In Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence.

[38] Edith Law and Haoqi Zhang. 2011. Towards Large-Scale Collaborative Planning: Answering High-Level Search Queries Using Human Computation. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, Wolfram Burgard and Dan Roth (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3675

[39] Ariel Levy, Monica Agrawal, Arvind Satyanarayan, and David Sontag. 2021. Assessing the Impact of Automated Suggestions on Decision Making: Domain Experts Mediate Model Errors but Take Less Initiative. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. 1–13.

[40] Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. 2021. Jurassic-1: Technical Details And Evaluation. Technical Report. AI21 Labs.

[41] Rensis Likert. 1932. A technique for the measurement of attitudes. Archives of psychology (1932).

[42] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. 2010. Turkit: human computation algorithms on mechanical turk. In Proceedings of the 23nd annual ACM symposium on User interface software and technology. 57–66.

[43] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What Makes Good In-Context Examples for GPT-3? arXiv preprint arXiv:2101.06804 (2021).

[44] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J. Cai. 2020. Novice-AI Music Co-Creation via AI-Steering Tools for Deep Generative Models. In CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020, Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik (Eds.). ACM, 1–13. https://doi.org/10.1145/3313831.3376739

[45] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. arXiv preprint arXiv:2104.08786 (2021).

[46] Päivi Majaranta and Kari-Jouko Räihä. 2007. Text entry by gaze: Utilizing eye-tracking. Text entry systems: Mobility, accessibility, universality (2007), 175–187.

[47] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2021. Cross-Task Generalization via Natural Language Crowdsourcing Instructions. *arXiv preprint arXiv:2104.08773* (2021).

[48] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 438–448.

[49] Hegde Narayan, Jason D Hipp, Yun Liu, Michael Emmert-Buck, Emily Reif, Daniel Smilkov, Michael Terry, Carrie J Cai, Mahul B Amin, Craig H Mermel, et al. 2019. Similar image search for histopathology: SMILY. *NPJ Digital Medicine* 2, 1 (2019).

[50] Besmira Nushi, Ece Kamar, Eric Horvitz, and Donald Kossmann. 2017. On Human Intellect and Machine Failures: Troubleshooting Integrative Machine Learning Systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 1017–1025. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/15032

[51] Joe O'Connor and Jacob Andreas. 2021. What Context Features Can Transformer Language Models Use? *arXiv preprint arXiv:2106.08367* (2021).

[52] Changhoon Oh, Jungwoo Song, Jinhan Choi, Seonghyeon Kim, Sungwoo Lee, and Bongwon Suh. 2018. I Lead, You Help but Only with Enough Details: Understanding User Experience of Co-Creation with Artificial Intelligence. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). ACM, 649. https://doi.org/10.1145/3173574.3174223

[53] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.06732

[54] Daniela Retelny, Michael S Bernstein, and Melissa A Valentine. 2017. No workflow can ever be enough: How crowdsourcing workflows constrain complex work. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–23.

[55] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7.

[56] Daniel Rough, Keith Vertanen, and Per Ola Kristensson. 2014. An evaluation of Dasher with a high-performance language model as a gaze communication method. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*. 169–176.

[57] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[58] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 4222–4235. https://doi.org/10.18653/v1/2020.emnlp-main.346

[59] Alison Smith-Renner, Ron Fan, Melissa Birchfield, Tongshuang Wu, Jordan Boyd-Graber, Daniel S. Weld, and Leah Findlater. 2020. *No Explainability without Accountability: An Empirical Study of Explanations and Feedback in Interactive ML*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376624

[60] Ben Swanson, Kory Mathewson, Ben Pietrzak, Sherol Chen, and Monica Dinalescu. 2021. Story Centaur: Large Language Model Few Shot Learning as a Creative Writing Tool. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Online, 244–256. https://www.aclweb.org/anthology/2021.eacl-demos.29

[61] Bowen Tan, Zichao Yang, Maruan Al-Shedivat, Eric Xing, and Zhiting Hu. 2021. Progressive Generation of Long Text with Pretrained Language Models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 4313–4324. https://www.aclweb.org/anthology/2021.naacl-main.341

[62] Jaime Teevan, Shamsi T Iqbal, Carrie J Cai, Jeffrey P Bigham, Michael S Bernstein, and Elizabeth M Gerber. 2016. Productivity decomposed: Getting big things done with little microtasks. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 3500–3507.

[63] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. LaMDA: Language Models for Dialog Applications. *ArXiv preprint* abs/2201.08239 (2022). https://arxiv.org/abs/2201.08239

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008.

[65] Vasilis Verroios and Michael S Bernstein. 2014. Context trees: Crowdsourcing global understanding from local views. In *Second AAAI Conference on Human Computation and Crowdsourcing*.

[66] Cunxiang Wang, Boyuan Zheng, Yuchen Niu, and Yue Zhang. 2021. Exploring Generalization Ability of Pretrained Language Models on Arithmetic and Logical Reasoning. *arXiv preprint arXiv:2108.06743* (2021).

[67] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903* (2022).

[68] Sean Welleck, Ilia Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. 2020. Consistency of a Recurrent Language Model With Respect to Incomplete Decoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 5553–5568. https://doi.org/10.18653/v1/2020.emnlp-main.448

[69] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. *PromptChainer: Chaining Large Language Model Prompts through Visual Programming*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3491101.3519729

[70] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2021. Polyjuice: Generating Counterfactuals for Explaining, Evaluating, and Improving Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 6707–6723. https://doi.org/10.18653/v1/2021.acl-long.523

[71] Bingjun Xie, Jia Zhou, and Huilin Wang. 2017. How influential are mental models on interaction performance? exploring the gap between users' and designers' mental models through a new quantitative method. *Advances in Human-Computer Interaction* 2017 (2017).

Primitive	Online demos
Info, extraction (9)	plan extraction [55], arithmetic reasoning [70], Keyword-extract, airport-code-extract, contact-info, color scale extractor, read code and answer questions, Summarize restaurant reviews (AI21), table question answering (AI21)
Classification (6)	hate speech detection [24], tweet-classifier, esrb rating, Automatically generating Request for Admissions, evaluate quiz answers, Classify news topics (AI21)
Rewrite (26)	program synthesis [6], Wordtune, generate database specific SQL code, parse-understructed-text, text to command, English to French, movie to emoji, tl;dr, sql-request, js-multi-line-to-one-line, js2python, html generation, description to app design description to todo list, Summarize-for-2nd-grade, Grammar-correction, third-person converter, rewrite as an attorney, Simplifying legal language, more polite, summarize famous people thoughts, speak in some personality, rewrite helper, mood color, De-jargonizer (AI21), Convert text to table (AI21)
Split points (1)	turn-by-turn directions
Composition (4)	Notes to summary review creator Description to ads Writing full emails from key points
Factual query (11)	add info to table, table computation, company to categories, factual answer, js-chatbot, ai-chatbot-tutor, sarcastic chatbot, imdb movie link, guess movie, Explain a word (AI21), Sports trivia (AI21)
Generation (8)	drafting email responses [66], Keyword 2 name, Generate poetry, spreadsheet generator, topic to horror story, Predict the outcome (AI21), project description generator (AI21), generate catchy headline (AI21),
Ideation (8)	scifi-booklist, essay outline, create study notes, interview questions, content creation for marketing, topic to quiz questions, VR fitness idea illustrator, blog post ideation (AI21)

Table 2: A survey of 73 online demos that inspired the design of our operation, mostly from published manuscripts, the OpenAI official GPT-3 example page, the AI21 tutorial, and the demo collection repository. All the links are last accessed in 2021/08.

A IDENTIFYING LLM PRIMITIVE OPERATIONS

We reviewed 73 existing demos to identify promising LLM capabilities that may help overcome the challenges above by scoping the inputs/outputs to be more amenable to what an LLM can handle.

First, we collected demos from LLM official websites (*e.g.,* GPT-3 and Jurassic), social media, and published case studies by searching for keywords including "GPT-3," "language model," "prompt," etc. After removing some demos that were highly open-ended rather than targeted (*e.g.,* generic chatbots), we iteratively sorted the demos into eight LLM primitive operations, as shown in Table 1. For example, we distinguished between operations that had different expected data mappings (one-to-many v.s. many-to-one), and different application types (deterministic v.s. creative). We then grouped the primitives into three high level groups based on which LLM challenge they may help address. The groups also appear to be consistent with categories presented on the GPT-3 tutorial page,[6] which highlighted typical NLP tasks like *Classification*, *Generation* (*i.e.,* gather additional information in Table 1b), *Transformation* (*i.e.,* re-organization). Finally, we further refined the primitive categories and names based on feedback from three pilot users (one LLM expert and two UX engineers with basic knowledge of LLM prompting).

B ADDITIONAL DETAILS FOR USER STUDY

B.1 Questions in the Exit Survey

After completing the given task in both conditions, participants self-rated their experience on the following dimensions, in the form of seven-point Likert scale [41]. Each question was asked twice, once on Sandbox and once on Chaining. They described their reasoning along with the ratings.

- *Match goal*: I'm satisfied with my final results from [Sandbox/Chaining]; they met the task goal.
- *Think through*: The [Sandbox/Chaining] system helped me think through what kinds of outputs I would want to complete the task goal, and how to complete the task.
- *Transparent*: The [Sandbox/Chaining] system is transparent about how it arrives at its final result; I could roughly track its progress.
- *Controllable*: I felt I had control creating with the [Sandbox/Chaining] system. I can steer the system towards the task goal.
- *Collaborative*: In [Sandbox/Chaining], I felt I was collaborating with the system to come up with the outputs.

Additionally, participants also answered the following two free form questions:

- *Difference*: What were the differences, if any, between the experience of completing the task using Sandbox and Chaining?
- *Vision*: If you were using language models in your work, in what situations would you prefer to use Sandbox? Chaining? Can you think of 1-3 concrete examples?
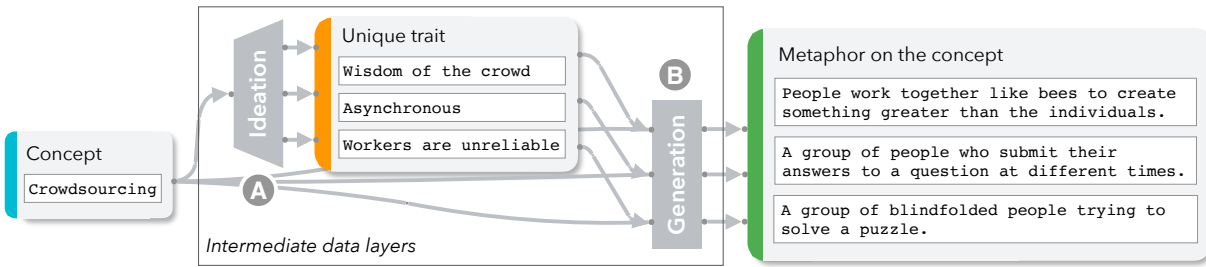
---

[6]https://beta.openai.com/docs/guides/completion/introduction

Figure 9: The pipeline for acronym expansion. The steps include: (A) An ideator that brainstorms various •*unique traits* for •*the concept* (crowdsourcing). (B) For each trait, a generator creates a related •*metaphor*.

B.2 Clickstream Categorization

we log the text status before and after each round of model run. Through sequence match, we recover what's generated by the model after each run, and how the participants edit the text in between of two runs. We split the logs into: (1) RUN the model, (2) UNDO the model, where people removed the generations from the previous run, making the resulting text more similar to prior to the previous run, (3) FORMAT, where people only add or remove line split or formatting-related stopwords, (4) CREATE-CONTENT, where people only insert meaningful spans to the text, (5) CURATE-CONTENT, where people make all the other kinds of refinements on the existing text — in Chaining, this is a merge of changing the instruction, prefix, and the data entries. We also logged (6) CHANGE-TEMPERATURE to denote when people make non-text based change on the model input, *i.e.,* temperature.

On top of the logs, we define *consecutive runs* (in Figure 6A) as those in which users did not change anything after the previous run (or only add formatting through line changes or adding stopwords, *i.e.,* RUN+FORMAT). Otherwise, the logs are counted as humans making *edits*.

B.3 Case 0: Metaphor Creation (Used in tutorial)

Description. Create metaphors for the concept of *crowdsourcing*, so that we can explain the *different aspects* of crowdsourcing in a poetic way. The pipeline is as in Figure 9.

A metaphor may look like:

> *In crowdsourcing, people are like bees; they work together to make honey.*

With the *concept* being "crowdsourcing", the *simile* being "bees", and the *similar aspect* being "work together."

Default baseline commands.

(1) In the form of question answering,

```
Question: What is a good metaphor for crowdsourcing?
Answer: a swarm of bees.
```

(2) In the form of command instruction,

```
Write a metaphor for the concept of crowdsourcing.
Concept: crowdsourcing
Metaphor: Crowdsourcing is like a game of chess. A crowdsourcer's skills, as in a chess player's skills, are combined with
another person's skills to make something new.
```

(3) List enumeration

```
The following is a list of metaphors on crowdsourcing.
1. Crowdsourcing is like a beehive -- Many people (bees) contribute to a larger cause.
```

(4) Few-shot example,

```
Concept: gratitude
Metaphor: gratitude is like a stream in that it's a force that can carry you along.
###
Concept: loss
Metaphor: loss is like a wing in that it's something you never wanted to lose, and it can take you away.
###
Concept: crowdsourcing
Metaphor:crowdsourcing is like a team sport in that it brings people to achieve one goal.
```

B.4   Case 1: Flashcard Creation

Description.   Suppose you will be traveling to Paris next week, and you would like to create flashcards to learn about some basic French so you can have basic conversations with local people whenever you are in a non-English speaking region. Your goal is to create flashcards that are both **diverse** and **personalized** to your travel desires.

A flashcard may look like:

> *English: Where is a good restaurant?; French: Où est un bon restaurant?*

Default baseline commands.   we offer three versions:

(1) In the form of question answering,

```
Question: What are some English and French sentence pairs useful for traveling to Paris?
Answers:
English: Where is a good restaurant?
French: Où est un bon restaurant?
```

(2) In the form of list enumeration,

```
The following are English and French sentence pairs useful for traveling to Paris.
1. Que fais-tu ici? / What are you doing here?
```

(3) Few-shot example,

```
English: I do not speak French.
French: Je ne parle pas français.
###
English: Where is a good restaurant?
French: Où est un bon restaurant?
###
English: Where's the bus station?
French: Où est la gare routière?
```

B.5   Case 2: Peer Review Paragraph Rewriting

Description.   The following is a peer review paragraph on Alex's presentation. Please rewrite it to be more **friendly and constructive**. The paragraph should include **1-3 concrete and actionable suggestions** for **each** of Alex's presentation problems.Original feedback:

> *Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents.*

An example output from the language model:

> *Alex, you have a lot of good ideas in your presentations. However, your slides are too cluttered with text. I would recommend that you use more graphics, pictures, and white space. Your slides should be simple enough that you could read your presentation off of the slides. This will allow you to engage the audience and not have to read from your notes. Also, you need to have a clear structure to your presentation. Lastly, you need to practice speaking. This will familiarize you with the material and reduce the likelihood of you meandering.*

Default baseline commands.   we offer two versions:

(1) End-to-end version,

```
Given the feedback, rewrite it into a friendly paragraph with concrete suggestions on each of Alex's presentation problems.
Feedback: Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from
topic to topic without a clear structure. He also does not engage with his audience when he presents.
Friendly paragraph: [LLM generation]
```

(2) Two-step version, where we query LLM for improvement suggestions first, and then ask it to integrate the problem and the suggestion.

```
Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to
topic without a clear structure. He also does not engage with his audience when he presents.
Give Alex some suggestions on his presentation:
1. [LLM generation]
Write one friendly paragraph that covers all the presentation problems and suggestions:[LLM generation]
```
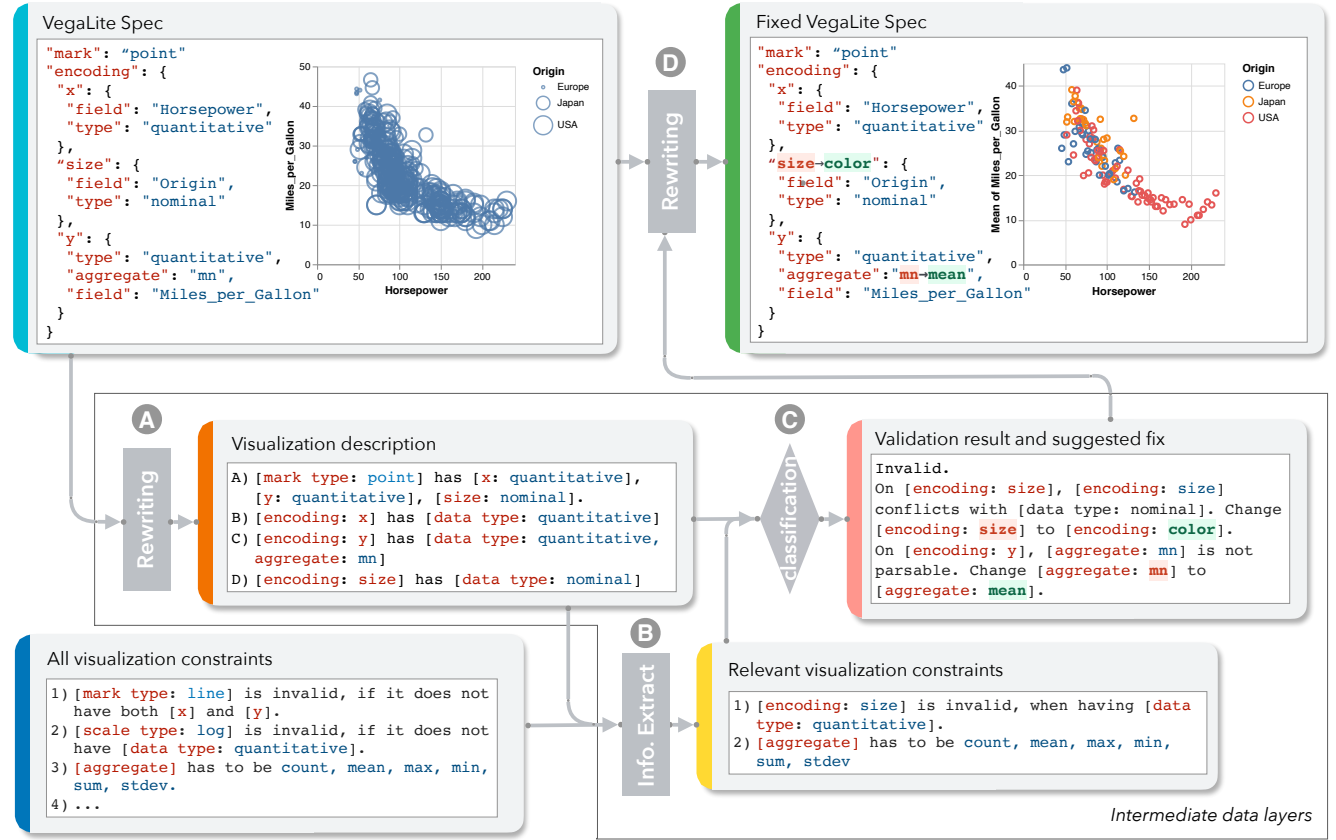
C FULL LLM CHAINS FOR CASE STUDIES

Figure 10: The LLM Chain for visualization bug fixing (in VegaLite). The stages include: (A) A Rewrite step that transforms the •_json format VegaLite spec_ into •_natural language description_, so to eliminate noise from data. (B) An Information Extraction step that locate •_related visualization rules_. (C) A Classification step that verifies the description as either •_valid or invalid_ (with concrete errors and fixes). (D) A Rewriting step that generates •_the fixed VegaLite spec_ based on the •_validity reasons_.
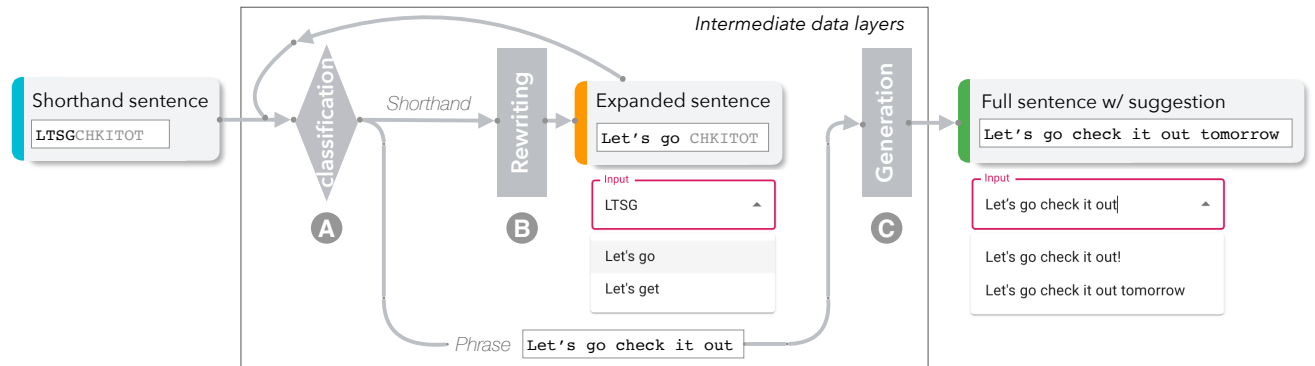

Figure 11: The LLM Chain for assisted text entry. The stages include: (A) A Classification steps that detects whether a •_given sentence_ •_contains a shorthand or not_. (B) If there exists certain shorthand, a Rewriting step expands it, so we arrive at the •_expanded sentence_ which can become the context for additional shorthand inputs. For "LTSG", it can be "Let's go" or "Let's get", which relies on human selection. (C) Otherwise, a Generation step autocompletes •_the sentence_.

D THE FULL IMPLEMENTATION OF PRIMITIVE OPERATIONS

Prompt template	Example	T
Classification: Assign the input based on limited categories. Most useful for branching logics and validation.		

Instruct `Classify if [detail-1] [detail-2].`
Input `[prefix-1]: (str)`
output `[prefix-2]: (str)`
	`Classify if the question is answerable.`
	`question: What is the square root of banana`
	`is answerable (Yes/No): No`
		0

(a) Primitive for examining the given input, to judge its value (potentially with reasoning), and what to do next.

Prompt template	Example	T
Information Extraction: Gather some information from the context.		

Instruct `Given [detail-1], extract [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: (string)`
	`Given text, extract airport codes for the cities.`
	`text: I want to fly from Los Angeles to Miami.`
	`airport codes: LAX, MIA`
		0.2

Rewriting: 1-1 mapping that changes the input to more machine-readable formats (e.g. json to natural language).		

Instruct `Rewrite [detail-1] into [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: (string)`
	`Rewrite the first-person text into third-person text.`
	`first-person text: I decide to make a movie`
	`third-person text: He decides to make a movie.`
		0.3

Split Points: 1-N mapping that is particularly useful for splitting contexts.		

Instruct `Split [detail-1] into a list of [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: 1.(list of strings)`
	`Split the descriptions on the direction into a list of turn-by-turn directions.`
	`Direction description: Go south on 95 until you hit Sunrise Blvd, then take it east to US-1 and head south.`
	`turn-by-turn directions: 1. Drive south on 95.`
	`2. Turn left onto Sunrise Blvd.`
	`3. Turn left onto US-1 SE.`
		0.3

Compose Points: N-1 mapping, the reverse operation of decomposition; merge multiple results back.		

Instruct `Write one [detail-1] to cover all the [detail-2].`
Input `[prefix-1]: (list of strings)`
output `[prefix-2]: (string)`
	`Write one review to cover all the restaurant name and notes.`
	`Restaurant name: The Blue Wharf`
	`Short notes: 1. Lobster great; 2.noisy; 3.service polite`
	`Review: The place is great if you like lobster. The noise level is a little high, but the service is polite.`
		0.5

(b) Primitives for reorganizing the given input, and re-format it by parsing and expressing them in different ways.

Prompt template	Example	T
Factual Query: Ask the model for a fact.		

Instruct `Given [detail-1], find [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: (string)`
	`Given the US state, find the population.`
	`US state: Washington`
	`Population: 7.6 million`
		0.3

Generation: Ask the model to do some creative "hallucination" on the input.		

Instruct `Given [detail-1], create [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: (string)`
	`Given the topic, create a two-sentence horror story.`
	`topic: Breakfast`
	`two-sentence horror story: He always stops crying when I pour the milk on his cereal. I just have to remember not to let him see his face on the carton.`
		0.7

Ideation: Ask the model for a list of ideas or examples.		

Instruct `Given [detail-1], the following is a list of [detail-2].`
Input `[prefix-1]: (string)`
output `[prefix-2]: 1.(list of strings)`
	`Given the interviewee, the following is a list of interview questions.`
	`Interviewee: A science fiction author`
	`Interview questions: 1. What's your favorite sci-fi book?`
	`2. Who inspired you to start writing books?`
		0.7

(c) Primitives for gathering additional clues from LLMs, when the desired output is too longer or too diverse.

Table 3: We design a list of primitive building blocks, each with default prompting templates and temperatures, and group them by their intended objectives. Examples are taken from https://beta.openai.com/examples.