LMCanvas: Object-Oriented Interaction to Personalize Large Language Model-Powered Writing Environments

Tae Soo Kim
taesoo.kim@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Arghya Sarkar
arghya@nyu.edu
New York University
New York, USA

Yoonjoo Lee
yoonjoo.lee@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Minsuk Chang*
minsuk.chang@navercorp.com
Naver AI Lab
Seongnam, Republic of Korea

Juho Kim
juhokim@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

ABSTRACT

Large language models (LLMs) can enhance writing by automating or supporting specific tasks in writers' workflows (e.g., paraphrasing, creating analogies). Leveraging this capability, a collection of interfaces have been developed that provide LLM-powered tools for specific writing tasks. However, these interfaces provide limited support for writers to create personal tools for their own unique tasks, and may not comprehensively fulfill a writer's needs—requiring them to continuously switch between interfaces during writing. In this work, we envision LMCanvas, an interface that enables writers to create their own LLM-powered writing tools and arrange their personal writing environment by interacting with "blocks" in a canvas. In this interface, users can create text blocks to encapsulate writing and LLM prompts, model blocks for model parameter configurations, and connect these to create pipeline blocks that output generations. In this workshop paper, we discuss the design for LMCanvas and our plans to develop this concept.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Generative AI, Large Language Models, Writing Support Tools, Object-Oriented Interaction

*Minsuk is now at Google.

1 INTRODUCTION

The advent of large language models (LLMs)—e.g., GPT-3 [4], GPT-NeoX [3], Jurassic-1 [15], LaMDA [20]—has transformed the writing process. Instead of manually drafting passages of text, writers can now hand over this effort to these models and almost instantly generate passages from an initial sentence or phrase. Beyond their generative capabilities, LLMs demonstrate significant few-shot and zero-shot performance [4] meaning that they are able to perform previously unseen tasks with only an instruction and/or a couple of examples—i.e., a prompt. By leveraging this ability of LLMs, writers can potentially automate or augment specific tasks in their workflows by using adequate prompts and, thus, further facilitate the writing process. For instance, based only on prompt examples provided for GPT-3 [17], writers can use LLMs to correct grammar, create an outline, produce analogies, or even change the point-of-view of a scene.

To seize the opportunity presented by LLMs, an assortment of products and interfaces have been created that leverage these models to provide writers with specific tools that automate steps in their writing workflows. For example, tools such as WordTune [11] and NotionAI [12] provide editing buttons that the user can click after selecting text to automatically rewrite it, change its tone, summarize it, elaborate on it, etc. Additionally, a variety of LLM-powered copywriting tools [1, 7] have also been created that provide writers with a variety of template forms that they can fill to generate specific types of writing (e.g., video description or script, blog introduction, article headline). Similarly in academia, various interfaces have been designed to leverage LLMs to support specific tasks: generate various forms of figurative language [5], summarize a writer's writing [8], brainstorm and combine ideas [9], or propagate writing edits across a story [14].

While the proliferation of these LLM-driven tools means that various writing tasks can now be supported, the individual needs and challenges of writers might not be fulfilled by these tools. Due to their type of writing, their fluency with a language, or other factors such as their style and workflow, a writer may have specific needs and challenges during their writing process. However, while existing interfaces provide a general set of tools, they provide limited or no support for the writer to create their own tools to support their unique tasks. Further, an interface may not provide a comprehensive set of tools that supports all of the writer's tasks and, thus, the writer may need to constantly switch between multiple

interfaces to support their workflows. As a result, the writer needs to scatter and adapt their writing workflow across a variety of interfaces.

In this work, we envision a canvas-based interface that enables writers to create their own personalized LLM-driven tools and configure them into one cohesive writing environment. Inspired by object-oriented interaction [2, 6, 10, 22–24] and block-based programming [18], we present the design for **LMCanvas**, an interface that enables users to interact with text and model blocks to flexibly create and arrange LLM-powered tools. Through the interface, users can create text blocks to encapsulate both their writing and LLM prompts, keep drafts as separate blocks, and organize them in the canvas. By connecting text blocks to model blocks (i.e., blocks that represent a set of model configurations), users can create LM pipelines, tools, that generate outputs as text blocks based on the input text and model block. After creating a set of tools, the user can flexibly arrange them in the canvas to create a writing environment customized to their needs and preferences. In this workshop paper, we discuss our design of the envisioned LMCanvas and future work to develop this concept.

2 LMCANVAS: DESIGN CONCEPT

In our envisioned interface, **LMCanvas**, writers can create four types of objects in an infinite canvas: *text blocks*, *model blocks*, and *pipeline blocks*. All of these blocks can be flexibly moved, copy-pasted, deleted, and connected to each other. Below, we detail the specific interactions that we aim to support for each type of block.

2.0.1 Text Blocks. In the canvas, the writer can create text blocks, which are objects that the writer can type text into and edit. When writing with LLMs, text can represent different types of content: actual writing, prompts, examples for prompts, generated outputs, etc. By compartmentalizing text into modular blocks, our interface allows the writer to flexibly organize and structure these different forms of text in their writing environment. For example, the writer can use a text block as their main text editor, maintain text blocks on the side containing alternative versions for certain paragraphs, and
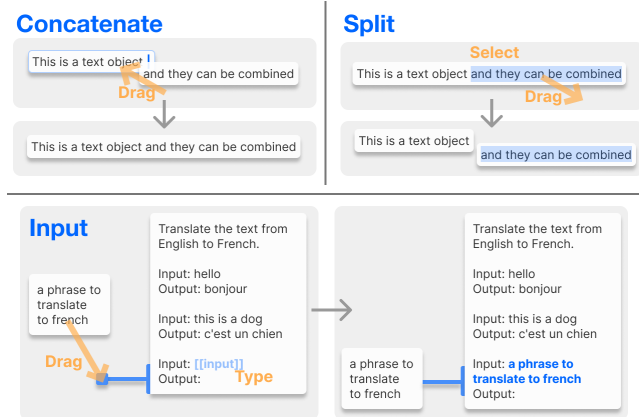
keep a text block with a prompt template to reuse when creating LLM-powered tools.

To support flexible use of text blocks, we design the following interactions specific to these blocks. **Resizing** allows the writer to change the format of text blocks for different types of usage (e.g., a larger block for a text editor) or to decrease their size to decrease clutter in the screen. When the writer decides that they do not have to keep two text blocks separate anymore (e.g., decided on the final versions for the first two verses of their poem), they can **concatenate** these blocks by drag-and-dropping one text block into the other. Alternatively, if the writer needs to modularize or separate certain parts of a text (e.g., to only draft one part of a paragraph), they can **split** off text by selecting it and dragging it outwards—creating a new text block. To allow writers to create reusable LLM-powered tools, the interface allows the user to create text blocks to which they can **input** other text blocks. Specifically, the user types the "[[input]]" command in a text block to create an "input prong". Then they can attach other blocks into this prong to replace the "[[input]]" command with the content of the attached text block. Finally, as writers may want writing support tools to act on selected text (e.g., generate metaphor for selected phrase or edit selected text to be shorter), the interface also allows users to create **select** blocks by typing the "[[select]]" command in a text block. The content of these blocks are replaced by any text that the user selects in the canvas.
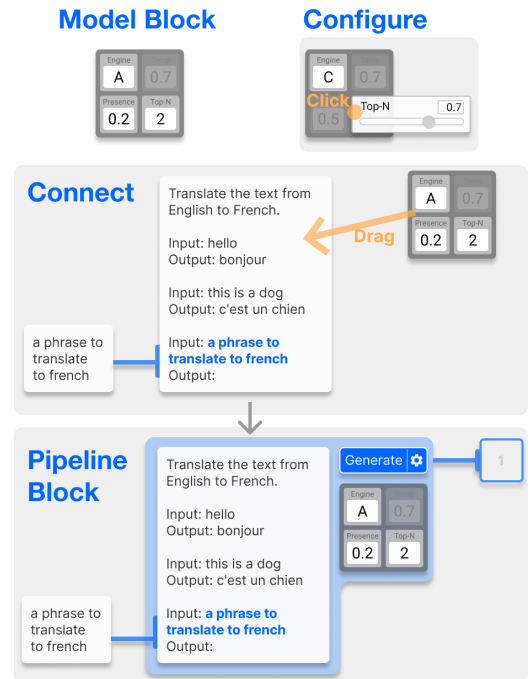
Figure 2: Model blocks represent a set of parameter configurations that the writer can configure, copy, and reuse. By connecting model blocks to text blocks, writers can create pipeline blocks that allow them to generate outputs based on the nested text and parameters.

Figure 1: Illustrations for the concatenate, split, and input interactions supported in text blocks.
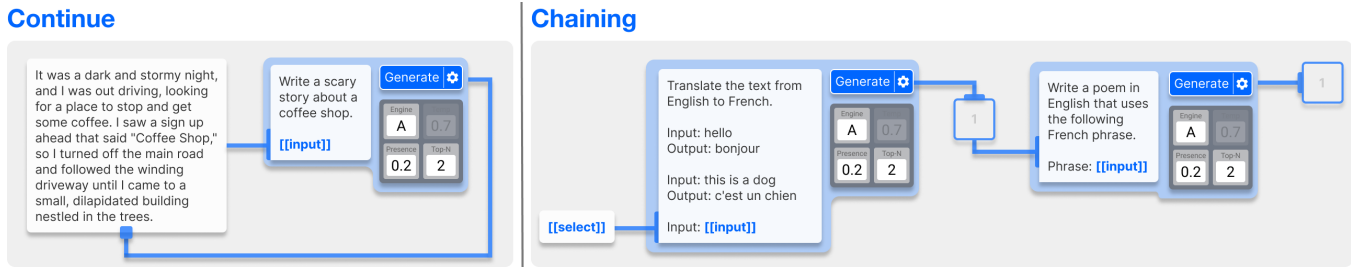
Continue

Chaining

Figure 3: The output container of pipeline blocks can be connected to text blocks to add generations as continuations, or to the input prongs of text blocks to chain pipelines.

2.0.2 Model Blocks. LLMs possess various parameters that control the generation process. For example, the temperature parameter determines the probability of the model generating more out-of-distribution or improbable text. Prior work has demonstrated that, when writing with LLMs, different configurations of these parameters can satisfy different user needs [13]. Thus, to support writers to set, test, and reuse parameter configurations, LMCanvas allows users to create multiple model blocks with different combinations of parameters (Figure 2). These blocks represents an instance of parameter configurations that the writer can re**configure** by clicking on a parameter and using the displayed widgets to change its value.

2.0.3 Pipeline Blocks. To generate text with the LLM, the user can **connect** a text block to a model block to create a pipeline block (Figure 2). When a writer clicks on "generate" in a pipeline block, the interface uses the nested text block as input and the model block as the parameter configurations to generate an output, which is presented as a text block. To test multiple inputs and parameter configurations, the writer can also **expand** a pipeline block by adding additional text and model blocks. In this case, when the pipeline block generates, it produces a generation for each pairing of text and model blocks inside the pipeline block.

By default, each time a pipeline block generates, it adds the generation as a text block in the output container—the box containing "1" that prongs out of the pipeline block as seen in Figure 2. However, through drag-and-drop, the writer can connect this output container to (1) a text block to add the generations from the pipeline as **continuations** to that text block (left in Figure 3), or (2) an input prong to **chain** multiple pipeline blocks together and create more complex tools [21] (right in Figure 3). Additionally, if the writer connects the output of a pipeline to a **select** block, the interface replaces any text selected across all text blocks in the canvas with the generation produced by the pipeline. These various forms of connecting pipeline blocks can enable the writer to create a variety of tools from the same basic blocks.

3 FUTURE WORK

In this workshop paper, we outlined the foundational objects and interactions that we aim to support in our envisioned LMCanvas. Our goal with this interface is to enable writers to more effectively leverage LLMs by personalizing their use to fit their unique workflows, needs, and challenges. At the current stage of the project, we have developed an initial prototype that supports the three

objects presented in this paper and their basic interactions. With this prototype, we are planning to conduct formative studies to understand the various tools that writers can create with LMCanvas, the benefits and drawbacks of the interfaces, and additional blocks and interactions that writers might need. Based on the findings, we plan to improve and expand on the concept. Beyond the directions for improvement to be distilled from the formative study, there are additional future directions that we plan to pursue with LMCanvas.

First, an additional benefit of representing writing as text blocks is that this can enable the interface to maintain a separate history for each text block. With this modularized history, writers can check on and revert changes for only specific parts of their writing, and they can also reflect back on their generation attempts by seeing what inputs and parameter configurations were previously used. We are planning to implement this modularized history for text blocks and to enable users to interact with it—e.g., dragging the text input that generated a text block out from the history and into the canvas.

Second, in future versions of LMCanvas, we aim to support various types of output containers for pipeline blocks. Currently, the prototype only supports containers that keep generated text blocks as a list. However, when dealing with a large quantity of generated outputs, writers may need alternative methods to look at and explore generated outputs. For example, generations could be encoded in a scatterplot [16] to enable the writer to visualize the output space.

Finally, identifying effective prompts (i.e., prompt engineering) is a major hurdle in leveraging LLMs. While tools have been designed to facilitate this in well-defined tasks where there is a "ground-truth" [19], there is limited work that investigated how to support prompt engineering in open-ended and more creative tasks. Through our initial versions of LMCanvas, we aim to investigate mechanisms to facilitate prompt engineering in open-ended writing and to incorporate these into the interface. For example, the interface could allow writers to drag-and-drop text blocks into pipeline blocks as positive or negative examples, and leverage these in the back-end to produce outputs more aligned to the writers' preferences.

ACKNOWLEDGMENTS

REFERENCES

[1] Jasper AI. 2022. *Jasper - AI Copywriter | AI Content Generator for Teams*. Retrieved Feb 23, 2023 from https://www.jasper.ai/

[2] Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands) *(CHI '00)*. Association for Computing Machinery, New York, NY, USA, 446–453. https://doi.org/10.1145/332040.332473

[3] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. (2022).

[4] 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. arXiv:2005.14165 [cs.CL]

[5] Tuhin Chakrabarty, Vishakh Padmakumar, and He He. 2022. Help me write a poem: Instruction Tuning as a Vehicle for Collaborative Poetry Writing. *arXiv preprint arXiv:2210.13669* (2022).

[6] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2016. Beyond Snapping: Persistent, Tweakable Alignment and Distribution with StickyLines. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 133–144. https://doi.org/10.1145/2984511.2984577

[7] CopyAI. 2022. *Copy.ai: Write better marketing copy and content with AI*. Retrieved Feb 23, 2023 from https://www.copy.ai/

[8] Hai Dang, Karim Benharrak, Florian Lehmann, and Daniel Buschek. 2022. Beyond Text Generation: Supporting Writers with Continuous Automatic Text Summaries. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[9] Giulia Di Fede, Davide Rocchesso, Steven P Dow, and Salvatore Andolina. 2022. The Idea Machine: LLM-based Expansion, Rewriting, Combination, and Suggestion of Ideas. In *Creativity and Cognition*. 623–627.

[10] Han L. Han, Junhang Yu, Raphael Bournet, Alexandre Ciorascu, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2022. Passages: Interacting with Text Across Documents. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 338, 17 pages. https://doi.org/10.1145/3491102.3502052

[11] AI21 Labs. 2022. *WordTune | Your personal writing assistant and editor*. Retrieved Feb 23, 2023 from https://www.wordtune.com/

[12] Notion Labs. 2022. *Notion AI*. Retrieved Feb 23, 2023 from https://www.notion.so/product/ai

[13] Mina Lee, Percy Liang, and Qian Yang. 2022. CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. *CoRR* abs/2201.06796 (2022). arXiv:2201.06796 https://arxiv.org/abs/2201.06796

[14] Yoonjoo Lee, Tae Soo Kim, Minsuk Chang, and Juho Kim. 2022. Interactive Children's Story Rewriting Through Parent-Children Interaction. In *Proceedings of the First Workshop on Intelligent and Interactive Writing Assistants (In2Writing 2022)*. 62–71.

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

[16] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. *Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173943

[17] OpenAI. 2022. *Examples - OpenAI API*. Retrieved Feb 23, 2023 from https://platform.openai.com/examples/

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[19] Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M. Rush. 2022. Interactive and Visual Prompt Engineering for Ad-hoc Task Adaptation with Large Language Models. https://doi.org/10.48550/ARXIV.2208.07852

[20] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. LaMDA: Language Models for Dialog Applications. *CoRR* abs/2201.08239 (2022). arXiv:2201.08239 https://arxiv.org/abs/2201.08239

[21] Tongshuang Wu, Michael Terry, and Carrie J. Cai. 2021. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. *CoRR* abs/2110.01691 (2021). arXiv:2110.01691 https://arxiv.org/abs/2110.01691

[22] Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-Oriented Drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4610–4621. https://doi.org/10.1145/2858036.2858075

[23] Haijun Xia, Bruno Araujo, and Daniel Wigdor. 2017. Collection Objects: Enabling Fluid Formation and Manipulation of Aggregate Selections. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5592–5604. https://doi.org/10.1145/3025453.3025554

[24] Haijun Xia, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor. 2018. *DataInk: Direct and Creative Data-Oriented Drawing*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173797