GenLine and GenForm: Two Tools for Interacting with Generative Language Models in a Code Editor

Ellen Jiang
ellenj@google.com
Google Research

Edwin Toh
edwintoh@google.com
Google Research

Alejandra Molina
alemolinata@google.com
Google Research

Aaron Donsbach
donsbach@google.com
Google Research

Carrie Cai
cjcai@google.com
Google Research

Michael Terry
michaelterry@google.com
Google Research

ABSTRACT

A large, generative language model's output can be influenced through well-designed *prompts*, or text-based inputs that establish textual patterns that the model replicates in its output [6]. These capabilities create new opportunities for novel interactions with large, generative language models. We present a macro system with two tools that allow users to invoke language model prompts as macros in a code editor. GenLine allows users to execute macros inline as they write code in the editor (e.g., "Make an OK button" produces the equivalent HTML). GenForm provides a form-like interface where the user provides input that is then transformed into multiple pieces of output at the same time (e.g., a description of web code is transformed into HTML, CSS, and JavaScript).

KEYWORDS

macros; generative models; prompt programming; code synthesis

ACM Reference Format:
Ellen Jiang, Edwin Toh, Alejandra Molina, Aaron Donsbach, Carrie Cai, and Michael Terry. 2021. GenLine and GenForm: Two Tools for Interacting with Generative Language Models in a Code Editor. In *The Adjunct Publication of the 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21 Adjunct), October 10–14, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3474349.3480209

1 INTRODUCTION

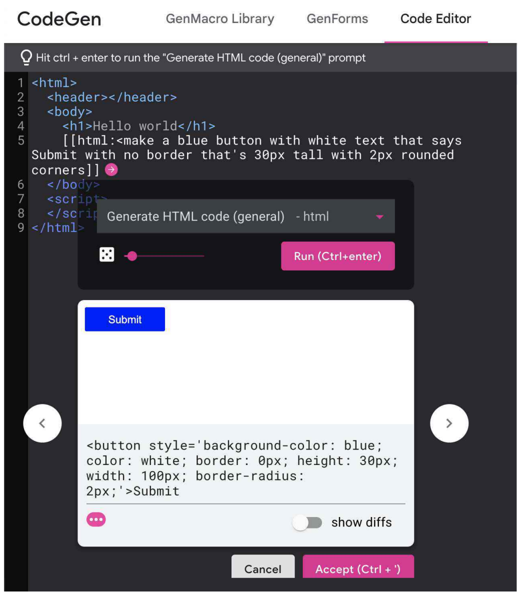
Generative models (e.g., [6, 8]) are designed to create plausible continuations of their input. For example, given the text, *"After work, I went to the"*, a model could then generate sentences or even paragraphs of text that resemble the start of a story.

A key finding of larger, more recent models such as GPT-3 [6] is that the input can be crafted in a way such that the model can then perform specific tasks. For example, one could prime the model to generate HTML given a natural language description, as in this modified example originally posted on Twitter [10]:

```
description: a red button that says stop
html: <button style = 'color: white;
    background-color: red'>Stop</button>
description: a blue button with white
    text that says Submit
html:
```

The model then follows this pattern and typically produces the output "<button style = 'background-color: blue; color: white;'> Submit</button>".

This type of guiding input (shown above) is often called a *prompt* [5, 6]. At the most basic level, a prompt is nothing more than the input text to the generative model. However, a number of compelling GPT-3 demos [6] demonstrate that prompts can be written to customize a single model to perform a wide range of tasks, such as transforming natural language instructions into fiction, source code,


CodeGen
GenMacro Library
GenForms
Code Editor
Hit ctrl + enter to run the "Generate HTML code (general)" prompt
Generate HTML code (general) - html
Run (Ctrl+enter)
Submit
show diffs
Cancel
Accept (Ctrl + ')

```
<html>
  <header></header>
  <body>
    <h1>Hello world</h1>
    [[html:<make a blue button with white text that says
Submit with no border that's 30px tall with 2px rounded
corners]]
  </body>
  <script>
  </script>
</html>
```

```
<button style='background-color: blue;
color: white; border: 0px; height: 30px;
width: 100px; border-radius:
2px;'>Submit
```

Figure 1: GenLine provides a front-end interface to macros written using prompt programming of a generative language model. The inline user input to the model is demarcated using a double-bracket notation (e.g., "[[Make an OK button]]"). Users can choose which macro to run in the drop-down menu. Model output appears at the bottom of the dialog box and is editable before insertion into the document.

or SVG graphics (among many other types of output) [1, 2, 5, 9]. Defining and designing the prompt is referred to as *prompt programming* [5, 6]. Prompt programming creates new opportunities for novel end-user interactions with large, generative language models.

The relatively small size of prompt programs brings to mind more traditional scripts or macros, such as those written in a scripting language or within an application (e.g., a word processor or software development environment). Extending this analogy further, it is not difficult to imagine parameterizing a prompt in such a way that end-users can use it as a macro or a script. For example, a macro author could create a prompt that takes natural language descriptions as input, and generates HTML as output (as in the example above).

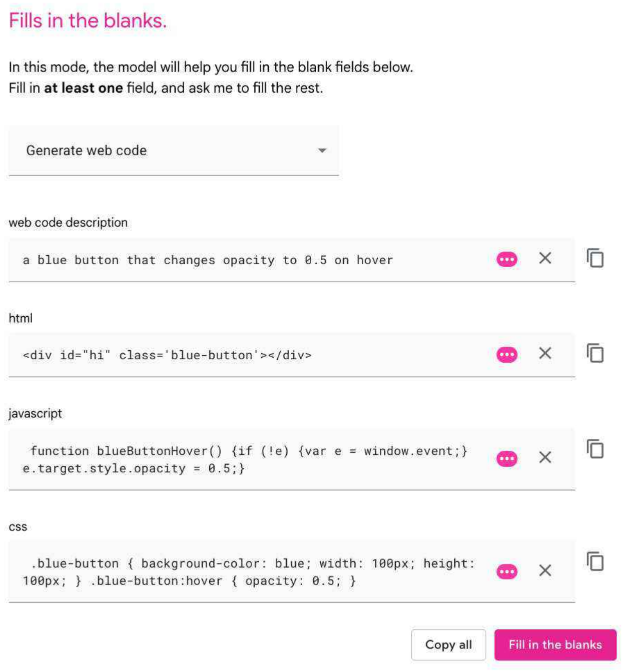
We demonstrate this concept of generative language macros (or "genmacros") in the context of a code editor. Specifically, we introduce a pair of tools, **GenLine** and **GenForm**, that provide first-class interface support for transforming natural language requests into code within a code editor. Our tools make use of prompt programming with a version of the model [7] described in [3] to synthesize code from natural language descriptions. The model was not specifically trained to support software development, though its corpus does include both natural language and code, and has been evaluated for its code generation capability [4].

2 GENLINE

The GenLine tool (Figure 1) allows users to pass inline requests to a genmacro using a double-bracket notion (e.g., "[[html: Make an OK button]]"). When GenLine is invoked, a dialog box appears with the model's output together with a rendered preview. We have produced genmacros for converting natural language requests to HTML, JavaScript, and CSS. To choose which genmacro to invoke, the user can prepend a keyword for the macro to their request (e.g., "html:", "css:"), or choose the desired genmacro from a drop-down menu in the GenLine dialog box (Figure 1).

GenLine is conceptually similar to a traditional autocomplete mechanism, but because it is backed by a generative language model, it can accommodate relatively flexible input. For example, a macro user can mix natural language instructions with code, allowing the user to specify in natural language how they want to modify the code (e.g., to change a button's height: "[[css: Make this 30 px tall <button>OK</button>]]"). We call this interaction pattern *mixed inputs* to capture the notion that the model enables the user to mix conceptually different types of input in their request (e.g., mixing code and natural language).

GenLine provides explicit functionality to address common issues found in the output of generative language models, such as the potential for it to generate unneeded content, content with small errors, or unusable content. Specifically, GenLine provides controls that allow end-users to 1) navigate multiple alternatives produced by the model, 2) edit model output prior to inserting it into the editor, and 3) transition to external resources when the model does not produce the desired output (by performing a web search based on the user's input).


Fills in the blanks.

In this mode, the model will help you fill in the blank fields below.
Fill in **at least one** field, and ask me to fill the rest.

Generate web code

web code description

```
a blue button that changes opacity to 0.5 on hover
```

html

```
<div id="hi" class='blue-button'></div>
```

javascript

```
 function blueButtonHover() {if (!e) {var e = window.event;}
e.target.style.opacity = 0.5;}
```

css

```
 .blue-button { background-color: blue; width: 100px; height:
100px; } .blue-button:hover { opacity: 0.5; }
```

Figure 2: GenForm provides a structured interface for generative language macros that can produce multiple outputs. In this example, the user provides a natural language description of the desired web code and the model fills in the corresponding HTML, JavaScript, and CSS.

3 GENFORM

While GenLine produces a single type of content (e.g., HTML code), GenForm (Figure 2) takes advantage of a model's ability to produce multiple types of content simultaneously. As an example, Figure 2 shows how the user can describe the desired web content and functionality ("a blue button that changes opacity to 0.5 on hover"). When the genmacro is run, GenForm will fill in the HTML, JavaScript, and CSS fields. We term this interaction pattern *mixed outputs*, as a generative model enables the user to specify a high-level goal that is transformed into multiple, interrelated outputs, where these outputs may be of different forms (e.g., a description of a user interface component that is transformed into the necessary HTML, JavaScript, and CSS for that component).

GenLine and GenForm use few-shot examples for their prompts [5]. GenForm's prompts include multiple named fields to generate the form's content (e.g., "web code description: <natural language request> html: <html> css: <css> javascript: <js>").

REFERENCES

[1] 2021. OpenAI API: Code Completion. https://beta.openai.com/?app=productivity&example=4_4_0. Accessed: 2021-03-30.
[2] 2021. OpenAI API: Natural Language Shell. https://beta.openai.com/?app=productivity&example=4_2_0. Accessed: 2021-03-30.
[3] Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. 2020. Towards a Human-like Open-Domain Chatbot. arXiv:2001.09977 [cs.CL] Accessed: 2021-08-12.

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.CL] Accessed: 2021-08-13.
[5] Gwern Branwen. 2020. GPT-3 Creative Fiction. https://www.gwern.net/GPT-3. Accessed: 2021-03-30.
[6] Tom Brown et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.
[7] Eli Collins and Zoubin Ghahramani. 2021. LaMDA: our breakthrough conversation technology. https://blog.google/technology/ai/lamda/ Accessed: 2021-07-14.
[8] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2019. Music Transformer. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJe4ShAcF7
[9] Fabin Rasheed. 2020. Tweet. https://twitter.com/fabinrasheed/status/1284052438392004608. Accessed: 2021-03-30.
[10] Sharif Shameem. 2020. Tweet. https://twitter.com/sharifshameem/status/1282692481608331265. Accessed: 2021-04-07.