

Genre Editor: Word Processing with Generative Capabilities

Nishant Shankar
nshankar@cs.stonybrook.edu
112670702

Rishabh Khot
rkhot@cs.stonybrook.edu
112682983

Once upon a midnight dreary,

I had the misfortune of visiting a bar on the planet Venus. It was one of those rare occasions when you are just out
the sky was dark and gray and cold. The moon was a beautiful pale white with moonlight shining in the sky

Figure 1: A screenshot from our implementation showing predictive text options.

ABSTRACT

Recent advances in natural language processing have been adopted widely in various fields and products. Natural language generation in particular has been used to display real-time predictive text while typing on QWERTY keypads, search engines, and even email. In this project, we utilize recent progress on topic specific language generation to develop a proof of concept implementation of a word processor with predictive capabilities. We also present a heuristic evaluation of our system, and propose modifications based on user feedback we gathered. The [code](#) of Genre Editor is open source.

KEYWORDS

human computer interaction, natural language processing, word processor, interactive systems, text generation, heuristic evaluation

1 INTRODUCTION

Text editing has a rich history in human computer interaction research [23]. Text editors deal with manipulating plain text, and don't offer visual formatting options which are present in rich text. Word processors, a more feature laden and extensible version of rich text editors started gaining popularity in the 1970s, developing independently from computer research [11]. Interestingly, even in 1986, automatic spell checking and correction was considered a standard feature in most word processors. More sophisticated algorithms started being used to perform grammar checking and statistical machine translation (limited to words and simple phrases). As time moved on, word processors have started adopting emerging language technologies such as neural machine translation [13] and AI-based text rewriting [8].

Different avenues for application of these technologies to editing text have also been explored, namely predictive text, and they also have considerable historical precedent. Smart typing in mobile phone keyboards are the spiritual successors of early predictive text in phone keypads such as T9 [22]. Media histories have been written on how autocomplete technologies conceptualize the notion of prediction [16], and recently Google has introduced Smart Compose

as an extension to email systems [4]. It's not an overstatement to say that predictive text is now a familiar technology in the public consciousness.

There is now increased interest in integrating language technologies into editors not just for day-to-day utility functions, but as a creative aid for writers. Much of it has been popularized in the press [19] [1] [14], but some herald new models like GPT-2 to be the new authors of tomorrow. This framing of language models as being creative sources of their own is not grounded in the realities of these models. More often than not, models tend to degenerate when producing longer texts, and cannot maintain themes across document-sized texts. Instead, using these models to augment the writing process, with authors interacting with language models to improve upon shorter suggestions seems to be more useful for writers.

To this end, we propose to create a rich text editor that integrates a language model to offer predictions similar to Smart Compose. In our formulation, the user will instead select some part of the text which is used as the prompt for the generation. Details will be discussed in later parts of the paper, but here we must mention another major change to the generation from technologies like Smart Compose and Autocomplete. Instead of just basing the predictions on the prompt alone, we also offer the user choice to select a particular genre. Genre refers to a categorization of similarities of form, style, subject across different texts, and to say a text is in a particular genre is to recognize unique characteristics it possesses. Having said that, generating short form text according to genre isn't that feasible, as genre is not typified in shorter contexts. It is more appropriate to say the generation is guided by topics and words commonly used in some genres, but we take the liberty to call our implementation Genre Editor.

The rest of this paper is organized as follows: in section 2 we discuss related work from both an HCI and NLP perspective. In section 3, we go deeper into the particular NLP model we chose to use for our work. In section 4, we describe the system architecture, and present the design of the user interface. In section 5, we perform a heuristic evaluation of the system and present a small user study we performed. In section 6, we propose future work based on our evaluation. We conclude the paper in section 7.

2 RELATED WORK

Design in word processors has been explored, but most of it involves design of experimental and somewhat niche features, as some have noted that word processor design has mostly stagnated in vision [20]. Crowdsourced editing enabled through Mechanical Turk has been shown to be a feasible way of creating interactive collaborative word processors [3]. There has been work on creating writing assistants integrated into conventional word processors that help contextualize written text [2]. Some really interesting design choices when displaying text options and interactively writing a story are seen in AI Dungeon [24].

Text generation has a long history, and neural text generation in particular has developed a dense history in the last five years. Work has been done on controlling and constraining the output of text generation using variational auto-encoders and attribute discriminators [7]. Transferring the style of a text to another and generating in such a manner has also been explored [15]. More complex generation models that use topic guided generation have also been developed utilizing different VAE architectures and Gaussian priors [21] [25]. We didn't decide to go ahead with these models as they involved relatively complex (or time consuming) training procedures, and not enough extensibility.

There has been considerable work in story generation in machine learning research. One approach is hierarchical generation: the model generates a prompt, and transforms it into a passage of text [6] [26]. Other approaches decompose input sentences into detected events, and then proceed to generate predictions based on these intermediate representations [17]. Though these approaches are interesting ways to generate stories in the long form, we focus on shorter form generation. And they only condition based on a prompt, not on any other attributes.

Another way to address the problem of generation has been to fine-tune massively large language models trained on a huge text corpus. GPT-2 (1.5B parameters) is one of many such Transformer models used for text generation purposes [18]. Access to GPT-3, its successor, has been granted to various researchers and developers, and it has found a myriad number of uses [9]. CTRL is also a large

(1.63B parameters) Transformer model, but departs from GPT-2 in that it offers control codes: a way to specify style or content of the generated text [10]. This is closer to what we want, but it is still a large and unwieldy model to use in a word processor.

We instead use a technique that leverages a pretrained language model and an added attribute classifier that guides generation [5]. This work is simple to use, does not require model training, and the topics can be customized by adding new wordlists.

3 MODEL DETAILS

UberAI developed an alternative to extensive fine-tuning: simple attribute models (classifiers) based on user-defined wordlists. They refer to this model as PPLM (Plug and Play Language Models). The sampling process involves a forward and backward pass where gradients from this attribute model are used to change the hidden representations in the language model. In this case, the language model used is GPT-2. We will go over some important details of their approach in this section, but for more mathematical rigour we ask the reader to refer to the original paper [5].

We can get an understanding of the decoding process from Figure 2. Like in most language models, the first step involves a forward pass to compute likelihood. This likelihood is tied to which attribute we have picked, as the attribute model gives us the probability $\Pr(a | x)$. In the second step the hidden representations of the language model are updated by the backward pass. In the third step the updated representations and current token allow the creation of a new distribution over the vocabulary. Sampling from this updated distribution yields the next token in the sequence. The latent representations are updated at each time step, and the gradients thus are steered toward the desired attribute. Two additional normalizations are performed to ensure model fluency: minimizing KL divergence and post-norm Geometric fusion.

We will now briefly describe some non-trivial hyperparameters that are relevant to tuning PPLM's generation.

- **Step size:** This value influences the size of the step you take while performing the gradient update.

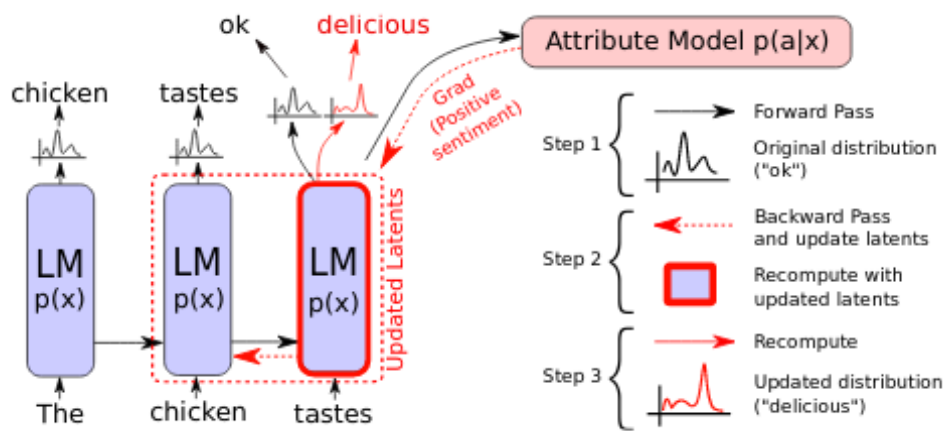


Figure 2: The generation process of PPLM as visualized in [5]. We present the figure here as is.

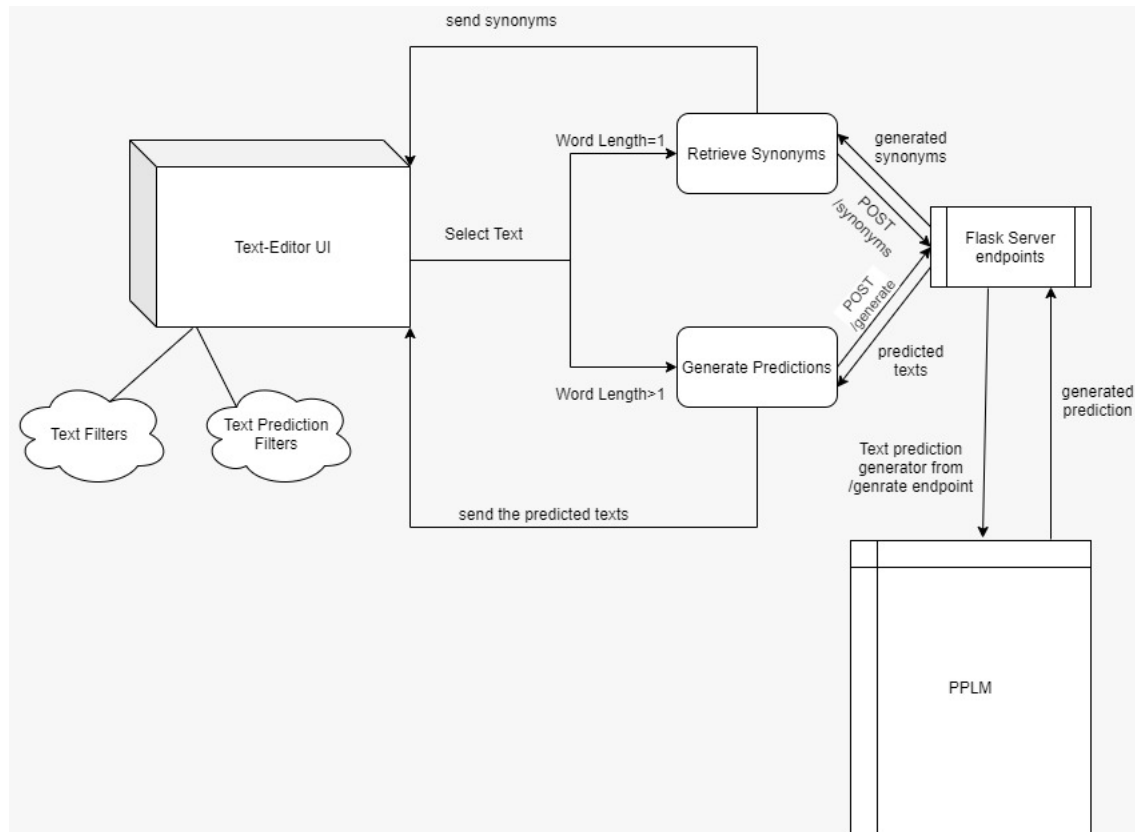


Figure 3: Flow diagram visualizing how Genre Editor works.

- **Temperature:** This controls randomness of predictions by scaling the logits before applying softmax. A temperature greater than 1 generates more "creative" responses whereas a lower temperature outputs generally expected predictions.
- **Top k:** A method of filtering sampled outputs where we keep only top k tokens with highest probability.
- **Horizon length:** Length of future to optimize over.
- **Window length:** Length of past which is being optimized; 0 corresponds to infinite window length.
- **Gamma:** The scaling coefficient for the normalization term in the gradient update.
- **GM Scale:** Scaling factor for the distributions in GM fusion.
- **KL Scale:** Scaling factor for the KL coefficient.
- **Repetition Penalty:** Penalize repetition. More than 1.0 -> less repetition.

4 SYSTEM DESIGN

Our application is built using HTML, CSS/Bootstrap, and Javascript. The Javascript is used to make the editor dynamic and communicate with Flask endpoints. Our particular application runs on Flask, but the design is such that the Flask portion can be rewritten as REST APIs that allow the front-end to be possibly decoupled from the back-end.

We now describe the entire front-end design process. The UI of Genre Editor has been created using HTML and CSS/Bootstrap. Initially, we used the `<iframe>` attribute for the input area, but we encountered various issues. The user wasn't able to see the predicted text inside the body of the editor, and the aesthetics of the application also suffered due to the `<iframe>`. Hence, we shifted to Bootstrap.

Bootstrap is a framework that helps in faster design and prototyping of web application front-ends. Bootstrap's responsive CSS adjusts to phones, tablets, and desktops and hence users will be benefited by utilizing it on any device. By keeping the basic word processor template in mind, we recreated features found in popular word processors so that users can comprehend and use it easily. We also incorporated radio buttons for genre selection instead of a drop-down. This allows faster selection of the desired option.

Users can use the Sticky Notes feature to scribble notes or jot down ideas before carrying it over to the finished draft. We utilize the right-hand side of the application for Sticky Notes. The user doesn't need to use an external notes tool as this UI clearly displays all the notes created till now.

Since the user might want to save the contents, a Save button is provided which saves the file in Docx format retaining all formatting. An upload file button is also provided. The Word Counter allows the user to keep a check on how much they've written. For

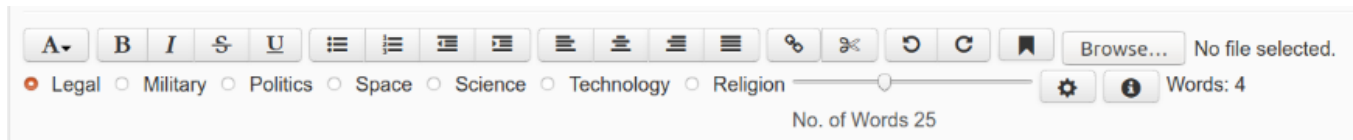


Figure 4: The design of the toolbar containing formatting options, genre radio selection, and hyperparameter tuners.

writers who prefer to use keyboards rather than a mouse, we have included the common keyboard shortcuts for text styling.

Rather than making use of the `<textarea>` for entering the text, we have used a `<div>` tag. This helped us in adding new tags like the loader attribute which gives the status of the prediction and the tags containing the predicted texts. We created a separate JavaScript file wherein the editor gets called and the functions get executed. This helps in making the editor responsive.

This jQuery Bootstrap plugin turns any DIV into a content editor. Here are the key features:

- Automatically binds standard hotkeys for common operations on Mac and Windows
- Drag and drop files to insert images, support image upload
- Allows a custom built toolbar, no magic markup generators, enabling the web site to use all the goodness of Bootstrap, Font Awesome etc
- Uses standard browser features, no magic non-standard code, toolbar and keyboard configurable to execute any supported browser command
- Does not create a separate frame, backup text areas etc - instead keeps it simple and runs everything inline in a DIV
- Supports mobile devices

Let us go over the two Flask endpoints: `/synonyms` and `/generate`.

- **/synonyms**: We use nltk to retrieve the synonyms of the selected word. A POST request to this endpoint is sent everytime the user selects a word.
- **/generate**: Two AJAX POST requests are sent to this endpoint when the user selects text that contains atleast two words and presses the `[Alt]` key. The POST payload consists of the currently selected genre and the selected text. Inside the endpoint, the model takes this as input and proceeds to perform the updates as described in the previous section.

5 EVALUATION

5.1 Heuristic Evaluation

We use a set of heuristics to see if our design adheres to some general principles of good design. Let us describe these and evaluate our system in detail:

- **Visibility of System Status** - Our UI constantly displays how many words a writer has written. We also display a loader animation (Figure 4) to show that the model is predicting the text. Since `/generate` takes less than 10 seconds to respond, we are not showing percentage completion.

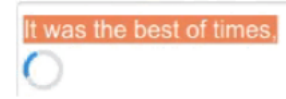


Figure 5: Loader animation.

- **Match system and Real World** - A person who has some experience with word processors can easily master Genre Editor. The symbols used on buttons are common norms for formatting of text and even the synonyms tag provides the user with a prompt to select a text to get the synonyms.
- **User control and freedom** - Since all the functionalities are present on the same page, a user needn't worry about the navigation of the page and all the close buttons are clearly specified on prompt
- **Consistency and standards** - As stated the document editor is following popular conventions followed by most word processors that people normally use and hence they don't have to learn new things for the same. We also integrate the standard popularised by sticky notes into the word processor.
- **Flexibility and efficiency of use** - Since each writer has a different way of handling a keyboard, we have created Genre Editor in such a way that a user can use keyboard for text editing or can use mouse to select options and use the same functions. A writer can use keyboard shortcuts for editing and hence there won't be any delay in navigating the system.
- **Aesthetic and minimalist design** - The UI showcases only those attributes which are required for a writer. We visually compartmentalize our design: the left part for showcasing of synonyms, the right part for making notes and the whole central part for writing actual content.
- **Help and documentation** - We provide an Info button that launches a popup describing in generic terms how the model works, and how the hyperparameters affect the text generation.

5.2 User Feedback

Now we discuss some user feedback we received on our application. At the time of writing the application hasn't been deployed, so we asked a few friends to use the application to write a paragraph. There were no particular constraints that we put on them, asking them only to keep in mind that they can use the genre selection to tune the text generation. Our aim here is to see how much the text generation feature is actually being used in the creative process.

As we can see from Table 1, there are different types of users that interact with the application. Some users like User 1 edit a lot, preferring to make sure the predictions are further refined

User	Total No. of Words	Own words	Predicted words	Example Output
1	114	22	92	It all started before the existence of time. It's an issue that is not as well discussed as some of our other problems, because there are a whole bunch of other science issues. Inflated balloon as Our universe is like an inflated balloon right now, and we will may not be able to reach the origin of everything. It is our intention to find a solution to all issues as we are a curious species in this world. The research about the existence of many theories of about the universe are everlasting. They are all based on the idea that the Universe consists of a similar pattern, a pattern that repeats itself on different levels. From atoms to galaxies, we can find that pattern around us.
2	117	49	68	There was only one way this could have happened: I was sitting on a chair on in my living room floor, waiting for the phone to ring. I got up, looked at the timer on the screen, and it read 6 seconds. I didn't know how I was going to get out of the house. I was going had gone for a walk around the neighborhood, but I didn't expect I would never go out again. Now the timer is broken, so I am stuck at the bottom of my house for power, wondering when I will be let out again. It seems like I have to be at work in the morning, but I don't think I will make it.
3	89	29	60	They walked toward the car, the first of what would later become hundreds. It was a black Mercedes sedan loaded with a heavy machine gun. They were merchants of death. It had been driven up to a checkpoint by two soldiers, one a lieutenant commander, another a lieutenant colonel. Their goal was to get the car past this checkpoint, and they would be golden. It was difficult to get around this area, and the road had been heavily damaged. So the drivers were forced to go through a series of detours.

Table 1: A sample from our study of how users interact with Genre Editor.

according to their stories. Users 2 and 3 take more time writing the paragraph, choosing to send more prediction requests. This means that they don't get a satisfactory prediction in the first go, but they still are open to suggestions.

User 2 just edited once to make a grammatical change, but overall contributed a lot of their own. This showcases the type of user that uses the prediction to explore new ideas, but not necessarily incorporate it in their own narrative. This user reported that the introduction of the "timer" and being "stuck at the bottom of the house" were two plot points that inspired an interesting new narrative hook that they could use moving forward.

User 3 didn't edit the predictions choosing to use them as is. But they spent the most time waiting for the perfect prediction. This also showcases a different type of user, one that is really enthusiastic about checking out as many different suggestions as possible. They said that the "the first of what would later become hundreds" and the fact that the car was "loaded with a heavy machine gun" drew some really interesting imagery of perhaps a large armed militia or organized crime network.

All the users noted that they tended not to switch genres within a single paragraph, which makes sense. Genre, as we noted, is more relevant in a longer form, but they did say that they switched genres once in a while to look at possible ways the plot could go.

A dominant thread from the user feedback (some of whom were not familiar with machine learning/computer science) was questioning of how the authorship attribution would happen. Who is the writer here, and can we say that the "AI" is "creative"? Well, we believe the writer here is the ultimate arbiter of how the text

is written. They have the ability to select which predicted text to include, if any, and how to edit it.

6 FUTURE WORK

Presently, we have used the wordlists included in [5]. Instead of these topics, it would be more relevant for writers if we used literary genres instead. This can be achieved by using existing book datasets like Project Gutenberg [12], pruning the books to obtain wordlists containing words that are unique or more frequent to a particular genre. Genres here could be Romance, Science Fiction and Fantasy, Horror to name a few.

The current system of displaying the synonyms uses the information obtained from nltk as is, without paying any attention to the context in which a particular word appears. It would be an interesting engineering challenge to use nearby word representations to develop some sort of heuristic that lets us rank the synonyms in order of relevance to the context.

Since text generation seems to be a valuable addition to a creative AI writing tool, other NLP technologies can also be added to the design. Summarization of large texts, rephrasing certain phrases in different ways, and pulling up semantically similar text from other books to avoid unintended plagiarism are just a few examples of the possibilities in developing exciting new creative writing technologies.

7 CONCLUSION

In this paper, we presented Genre Editor, a lightweight word processor with generative capabilities. We looked at various related

work in text generation, and how Plug and Play Language Models were extensible and fast enough to be used in our proof of concept implementation. We described the system design in detail, going over the front-end design and Flask endpoints. We presented a heuristic evaluation of the system, and based on the user feedback we obtained we conclude that our design helps in integrating predictions in such a way that the user tends to use them consistently. We see that users that edit their predictions don't try out multiple generation requests. Over time a user's writing process inculcates regular usage of the prediction feature. We conclude by suggesting ways in which more language technologies can be used as creative tools in the context of a word processor.

REFERENCES

- [1] Gregory Barber. 2019. Text-Savvy AI Is Here to Write Fiction. (2019). <https://www.wired.com/story/nanogenmo-ai-novels-gpt2/>
- [2] Alessio Bellino and Daniela Bascuñán. 2020. Design and Evaluation of WriteBetter: A Corpus-Based Writing Assistant. *IEEE Access* 8 (2020), 70216–70233.
- [3] Michael S. Bernstein, G. Little, R. Miller, B. Hartmann, M. Ackerman, D. Karger, D. Crowell, and Katrina Panovich. 2010. Soylent: a word processor with a crowd inside. In *UIST '10*.
- [4] M. Chen, B. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Y. Lu, Jackie Tsay, Yanan Wang, Andrew M. Dai, Z. Chen, T. Sohn, and Yonghui Wu. 2019. Gmail Smart Compose: Real-Time Assisted Writing. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019).
- [5] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, E. Frank, Piero Molino, J. Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. *ArXiv abs/1912.02164* (2020).
- [6] Angela Fan, M. Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. *ArXiv abs/1805.04833* (2018).
- [7] Zhiting Hu, Zichao Yang, Xiaodan Liang, R. Salakhutdinov, and E. Xing. 2017. Toward Controlled Generation of Text. In *ICML*.
- [8] Khari Johnson. 2020. Microsoft's AI-powered Editor can generate rewrite suggestions. (2020). <https://venturebeat.com/2020/03/30/microsofts-ai-powered-editor-can-generate-rewrite-suggestions/>
- [9] Aditya Joshi. [n.d.]. GPT-3 examples. ([n. d.]). <https://gpt3examples.com/>
- [10] N. Keskar, B. McCann, L. R. Varshney, Caiming Xiong, and R. Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *ArXiv abs/1909.05858* (2019).
- [11] Brian Kunde. 1986. A Brief History of Word Processing (Through 1986). (1986). <https://web.stanford.edu/~bkunde/fb-press/articles/wdprhist.html>
- [12] Shibamouli Lahiri. 2014. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Gothenburg, Sweden, 96–105. <http://www.aclweb.org/anthology/E14-3011>
- [13] Jennifer Langston. 2019. Microsoft's AI-powered Editor can generate rewrite suggestions. (2019). <https://blogs.microsoft.com/ai/mona-lisa-translation-research-products/>
- [14] J.D. Lasica. 2020. How Artificial Intelligence Can Help Authors Write a Better Novel. (2020). <https://www.writersdigest.com/write-better-fiction/how-artificial-intelligence-can-help-authors-write-a-better-novel>
- [15] Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, Retrieve, Generate: A Simple Approach to Sentiment and Style Transfer. *ArXiv abs/1804.06437* (2018).
- [16] X. I. Li. 2017. *Divination Engines: A Media History of Text Prediction*. Ph.D. Dissertation. <http://proxy.library.stonybrook.edu/login?url=https://www.proquest.com/dissertations-theses/divination-engines-media-history-text-prediction/docview/1964286735/se-2?accountid=14172>
- [17] Lara J. Martin, Prithviraj Ammanabrolu, W. Hancock, S. Singh, B. Harrison, and Mark O. Riedl. 2018. Event Representations for Automated Story Generation with Deep Neural Nets. In *AAAI*.
- [18] A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.
- [19] Sigal Samuel. 2019. How I'm using AI to write my next novel. (2019). <https://www.vox.com/future-perfect/2019/8/30/20840194/ai-art-fiction-writing-language-gpt-2>
- [20] Ernie Smith. 2016. It Makes No Sense That Word Processors Are Still Designed for the Printed Page. (2016). <https://www.vice.com/en/article/53dwb3/it-makes-no-sense-that-word-processors-are-still-designed-for-the-printed-page>
- [21] Hongyin Tang, M. Li, and Beihong Jin. 2019. A Topic Augmented Text Generation Model: Joint Learning of Semantics and Structural Features. In *EMNLP/IJCNLP*.
- [22] ValidConcept. 2009. T9: Text on Nine Keys. (2009). <http://www.validconcept.com/articles-t9.html>
- [23] Andries van Dam and David E. Rice. 1971. On-Line Text Editing: A Survey. *ACM Comput. Surv.* 3, 3 (Sept. 1971), 93–114. <https://doi.org/10.1145/356589.356591>
- [24] James Vincent. 2019. This AI text adventure game has pretty much infinite possibilities. (2019). <https://www.theverge.com/tldr/2019/12/6/20998993/ai-dungeon-2-choose-your-own-adventure-game-text-nick-walton-gpt-machine-learning>
- [25] W. Wang, Zhe Gan, H. Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and L. Carin. 2019. Topic-Guided Variational Autoencoders for Text Generation. *ArXiv abs/1903.07137* (2019).
- [26] Lili Yao, Nanyun Peng, R. Weischedel, Kevin Knight, Dongyan Zhao, and R. Yan. 2019. Plan-And-Write: Towards Better Automatic Storytelling. *ArXiv abs/1811.05701* (2019).

Settings

Tune the parameters to adjust the outputs as per the requirement. Check the help button for more information regarding the parameters and model understanding

Temperature 1

Number of Iterations (between 1 and 5):

3

Step Size:

0.03

Top K:

10

Grad Length:

10000

Horizon Length:

1

Window Length:

0

Gamma:

1.5

GM Scale:

0.9

KL Scale:

0.01

Repetition Penalty:

1

Close

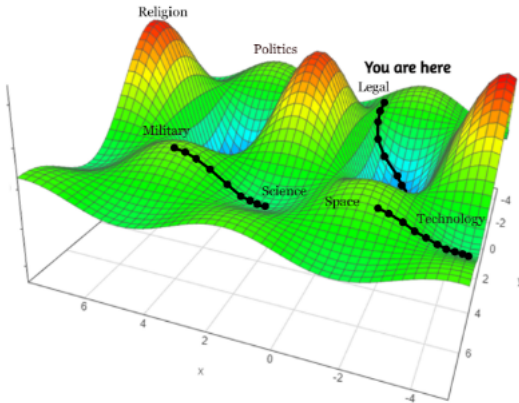
Figure 7: Hyperparameter tuner.

A SCREENSHOTS

Here we present some screenshots of our application.

Information

It is a common problem among writers to be faced with writer's block, and to have so many ideas in your head that none of them land on the page. Can we tackle this problem through software design and machine learning? We decided to integrate predictions into the editor. All the writer has to do is select the text that will act as a prompt and press a hotkey. Taking Google's autocomplete feature one step further, we provide multiple predictions based on the prompt in the editor's text area. This allows the writer to pick their poison, and get creative with the hallucinations of the machine learning model!



Temperature - used to control the randomness of predictions

Step Size - The gradient descent algorithm descends along a function by taking steps in the opposite direction of the gradient of that function

Window Length - The number of words to be used for processing

Repetition Penalty - Penalty Factor if same sentence gets predicted more than once

Close

Figure 6: Information popup.

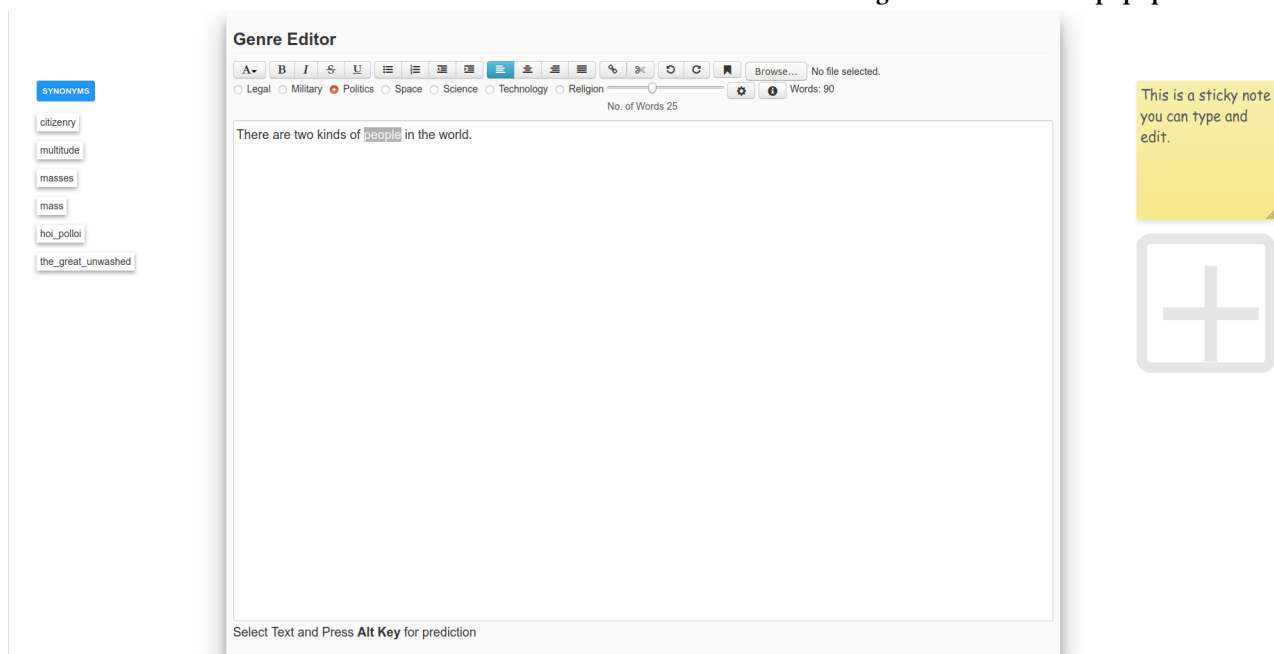


Figure 8: This is the general UI of Genre Editor.