UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title

Controlling Long-Form Large Language Model Outputs

Permalink

https://escholarship.org/uc/item/10h153t2

Author

Yang, Kevin

Publication Date

2023

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Controlling Long-Form Large Language Model Outputs

By

Kevin Yang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Dan Klein, Chair
Assistant Professor Alane Suhr
Assistant Professor Nanyun Peng

Fall 2023

Controlling Long-Form Large Language Model Outputs

Copyright 2023
by
Kevin Yang

Abstract

Controlling Long-Form Large Language Model Outputs

by

Kevin Yang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Dan Klein, Chair

As large language models have greatly increased in capability in recent years, it becomes increasingly important to improve our ability to exert control over their outputs. In this thesis, I discuss several such control schemes I have developed, ranging from pure inference-time control to finetuning-based alignment methods. I will first discuss highly general methods that apply to unstructured natural language generation, including both an inference-time control scheme called FUDGE [164] as well as a reinforcement-learning based finetuning approach called RLCD [169]. I will next discuss more specialized methods that can be used for control in more structured domains such as molecule design, program synthesis, and semantic parsing [167, 163]. Finally, I will show how many of these ideas can be used in conjunction with structured planning via prompting to extend our control to much longer outputs—in the range of thousands of words—in an automatic story generation application [168, 166].

Contents

Contents **i**

List of Figures **ii**

List of Tables **v**

1 Introduction **1**
1.1 Background 1
1.2 Contributions 2

2 Control For Natural Language **5**
2.1 Future Discriminators for Generation (FUDGE) 5
2.2 Reinforcement Learning from Contrastive Distillation (RLCD) 19
2.3 Additional Projects 31

3 Control For Structured Domains **32**
3.1 Stochastic Iterative Target Augmentation 32
3.2 Additional Projects 46

4 Long-Form Story Generation **48**
4.1 Recursive Reprompting and Revision (Re^3) 48
4.2 Detailed Outline Control (DOC) 65
4.3 Additional Projects 78

5 Conclusion **80**

Bibliography **82**

List of Figures

2.1 Illustration of one decoding step in FUDGE, for an example where the desired attribute a is formality. A large pretrained model $\mathcal{G}$ (dark blue) outputs unconditioned probabilities. Our binary predictor (red) predicts whether the eventual completed sequence will be formal for each possible continuation (computed for each candidate x_3, e.g., "want"; holding a fixed). The probabilities for each x_3 are multiplied (purple) and then renormalized to obtain $P(x_3|x_{1:2}, a)$, from which we sample the next token $x_3 =$"prefer." 7

2.2 Stylized illustration showing RLCD compared to standard RLAIF and context distillation on harmlessness attribute. RLCD generates preference pairs using two contrasting prompts p_+, p_-, and labels according to the prompt used, thus making use of both pairwise preferences for RL as well as directional attribute change in outputs as encouraged by prompts. RLCD then trains a preference model on the resulting pairs, which is used to guide the LLM alignment via PPO. 20

2.3 Illustration of PREADD applied to toxic output mitigation. PREADD obtains next-token probabilities $P(x_{i+1} \mid x_{1:i})$ for the original tokens $x_{1:i}$, as well as $P(x_{i+1} \mid r_{1:k}, x_{1:i})$ for $x_{1:i}$ prepended with an additional toxicity-encouraging prefix $r_{1:k}$. x_{i+1} is then sampled proportional to $P(x_{i+1} \mid r_{1:k}, x_{1:i})^\alpha P(x_{i+1} \mid x_{1:i})^{1-\alpha}$, with α set to a negative value to control against the toxicity encouraged by $r_{1:k}$. Meanwhile, probabilities of unrelated tokens (e.g., "and") are kept relatively unchanged. 31

3.1 Illustration of conditional molecular design. Molecules can be modeled as graphs, with atoms as nodes and bonds as edges. Here, the task is to train a translation model to modify a given input molecule into a target molecule with higher drug-likeness (QED) score. The constraint has two components: the output Y must be highly drug-like, and must be sufficiently similar to the input X. 34

3.2 Illustration of data generation process for conditional molecular design. Given an input molecule, we first use our generative model to generate candidate modifications, and then select sufficiently similar molecules with high property score using our external filter. In the unconditional setting where the model takes no input, we simply sample outputs from the model and filter by property score. . . 34

3.3 **Left**: QED test success rate vs. Chemprop predictor's RMSE with respect to ground truth. The red line shows the performance of the (unaugmented) VSeq2Seq baseline. **Right**: Same plot for DRD2. In each plot, the far left point with zero RMSE is obtained by reusing the ground truth predictor, while the second-from-left point is the Chemprop predictor we use to obtain our main results. Points further to the right are weaker predictors, simulating a scenario where the property is more difficult to model. 38

3.4 Cumulative number of unique training pairs seen by VSeq2Seq+ model after each augmentation epoch, on QED task. 40

3.5 **Left**: Epoch number vs. uniqueness, evaluated with the Chemprop proxy predictor, for VSeq-based models on QED dataset. VSeq+ and VSeq in blue and red respectively. **Right**: Same plot for DRD2. VSeq+ is trained without iterative target augmentation for the initial epoch 0, and trained with augmentation thereafter. 42

3.6 Illustration of our data generation process in the program synthesis setting. Given an input-output specification, we first use our generation model to generate candidate programs, and then select correct programs using our external filter. Images of input-output specification and the program A are from [16]. 42

3.7 A partially-solved example crossword puzzle from the 2021 American Crossword Puzzle Tournament, where our system won first place against 1,100 top human solvers. The highlighted fill KUNGFU answers the wordplay clue: *Something done for kicks?* 47

4.1 High-level overview of Re^3. 49

4.2 Illustration of Re^3's Plan module, which prompts a language model to generate a setting, characters, and outline based on the premise. Highlighting indicates generated text. 52

4.3 Illustration of the prompt constructed in Re^3's Draft module to generate each next story continuation. Our recursive reprompting approach combines pieces of the plan (blue) and previously generated story (grey) into a single prompt by concatenating the depicted components in order. 53

4.4 Re^3's Rewrite module reranks the Draft module's continuations for coherence and relevance. 55

4.5 Illustration of Re^3's Edit module. Starting from the Rewrite module's best continuation, we infer natural language facts about each character, and convert them to attribute-value pairs. New values (blue) are added to the attribute dictionary, and contradictory values (red) are corrected. 56

4.6 High-level overview of DOC. Our detailed outliner expands a brief initial outline into a more detailed outline. Our detailed controller then maintains faithfulness to the more detailed outline when drafting the main story. 66

4.7 Stylized example showing the main components of the structured prompt used to draft new story passages in Re^3 and DOC. Leveraging our detailed outline and detailed controller, new elements of DOC's prompt include character development over time (red), more detailed events based on outline leaf nodes (orange), future context (green), and improved setting and character information (purple). . . . 68
4.8 Diagram of new entry creation in the detailed outline. Our detailed outliner recursively expands outline items in breadth-first order. To create each new entry, it proposes candidate events and selects the best via filtering and reranking, and then detects the setting and relevant characters. 69

List of Tables

2.1 An example couplet by William Shakespeare. Every second syllable is stressed, following iambic meter, and the last words of each line (see/thee) rhyme. 9

2.2 Couplet completion results. Success (main metric), grammaticality, perplexity, and distinctness of different methods, tested on 154 prefix lines from Shakespeare sonnets. FUDGE substantially outperforms automated baselines on success and maintains high diversity, although quality unsurprisingly suffers compared to the base $\mathcal{G}$ due to the difficult constraint $\mathcal{F}$. Note Shakespeare's work is often "incorrect" due to the narrowness of our metric $\mathcal{F}$; he also scores poorly on text quality because our evaluation models are intended for more modern English. 10

2.3 Two examples of successful couplet completions (in purple) generated by FUDGE. 12

2.4 Topic control results. Success (main metric), grammaticality, perplexity, and distinctness for different methods. FINETUNE and WDEC often degenerate into repeating the given bag of words $\mathcal{W}$; this is ill-captured by perplexity, but results in poor grammaticality and distinctness. FUDGE substantially outperforms all baselines on success, including the strong gradient-based PPLM baseline, while preserving high quality and diversity. 13

2.5 Topic control human evaluations, pairwise comparisons. FUDGE achieves a substantially higher fraction of on-topic outputs compared to each baseline, in addition to higher average fluency (rated 1 to 5). 15

2.6 The first output from FUDGE when using the prefix "The issue focused on" for two topics. We use red to highlight words in the given bag of words $\mathcal{W}$ along with obvious forms (e.g., plurals), and cyan for other on-topic words, including related words not in the heldout bag $\mathcal{W}'$. More examples in [164] Appendix J. . 16

2.7 An example from the Fisher dataset. **Top:** The original Spanish transcription. **Middle:** The original English translation. **Bottom:** [126]'s more fluent version. 16

2.8 Machine translation formality results. BLEU (main metric) and average formality for different methods, with and without fine-tuning $\mathcal{G}$ on the Fisher domain. FUDGE increases the formality of translations compared to the base model $\mathcal{G}$ while preserving or increasing BLEU score. Conversely, $\mathcal{G}$ with style transfer overfits to the formality data, resulting in near-perfect formality but losing the original meaning. 18

2.9 Example translations by $\mathcal{G}$ (fine-tuned on the Fisher dataset) and FUDGE using the same $\mathcal{G}$. Original Spanish and [126] references also shown. In this setting, FUDGE achieves similar BLEU to $\mathcal{G}$ while increasing formality. While FUDGE often simply corrects punctuation or capitalization (top), it also makes more complex adjustments (bottom). More examples in [164] Appendix L. 18
2.10 An example conversation prefix and corresponding simulated preference data from RLAIF and RLCD using LLaMA-7B, where the task is to improve harmlessness. Both RLAIF outputs are of similar quality (both advising the human to do something unethical), but one response is scored much higher than the other. On the other hand, RLCD's o_+ is clearly preferable to o_- in this instance. 24
2.11 Human comparison results for RLCD against each baseline, evaluating harmlessness and helpfulness on harmlessness prompt set; helpfulness on helpfulness prompt set; and outline quality on story outlining prompt set. Annotators indicated which output was better, and by how much, on a 1-8 scale; scores here are normalized so that higher is better. RLCD is in all cases equal or better—often substantially better—compared to baselines, for all tasks and for preference data simulation at both 7B and 30B model scale. 26
2.12 Percentage of outputs preferred in GPT-4 binary evaluations when comparing RLCD to baselines for aligning LLaMA-7B. RLCD's outputs are substantially preferred by GPT-4 across all tasks and data simulation model scales, except comparing RLCD_{30B} to $\text{RLAIF}_{\text{30B}}$, where performance is relatively equal. . . . 27
2.13 Outputs generated by baselines and RLCD for an example from our helpfulness test set. All methods use the same LLaMA-7B during preference data simulation, but RLCD's final output is clearly more comprehensive in this case. 28
2.14 Average binary accuracy and probability for favoring gold human-preferred output on harmlessness and helpfulness data, for RLAIF and RLCD preference models. RLCD's preference models perform better on both datasets. 29
2.15 Percentage of outputs preferred in GPT-4 pairwise comparisons for RLCD vs. RLCD-Rescore variant (re-labeling outputs using RLAIF scoring prompts). RLCD dramatically outperforms RLCD-Rescore at 7B scale for preference data simulation, but rescoring becomes a viable alternative at 30B scale. 30

3.1 Performance of different models on QED and DRD2 conditional generation tasks. Italicized models with + are augmented by our algorithm. Best performance for each model architecture in bold, not including models that use additional unlabeled data. *Note that the semi-supervised and transductive settings for VSeq2Seq are not directly comparable to VSeq2Seq and VSeq2Seq+ due to using additional unlabeled data. However, they show that having access to such unlabeled inputs can substantially improve performance. But we emphasize that iterative target augmentation remains critical to performance in these settings: augmentation without an external filter instead decreases performance. 38

3.2 Ablation analysis of filtering at training and test time. "Train-Aug" indicates a model whose training process uses self-generated candidates to augment the data, while "Train+" is a model that additionally filters these candidates using the proxy according to our framework. "Test+" indicates a model that filters outputs at prediction time using the learned proxy predictor. We emphasize that the ground truth predictor is used only for final evaluation. The evaluation for VSeq2Seq(no-filter) is conducted after 10 augmentation epochs, as the best validation set performance only decreases over the course of training. 39
3.3 Performance of different models on QED and DRD2 unconditional generation tasks. VSeq+ is our full augmented model. 42
3.4 Model performance measured by top-1 generalization accuracy on Karel program synthesis task. MLE+ is our augmented version of the MLE model [16], while MLE + RL + Beam Search is their reinforcement learning method applied to the same architecture. 44

4.1 Comparison of RE^3 against two baselines, ROLLING and ROLLING-FT, in two separate experiments examining comparisons of interestingness, coherence, relevance, humanlikeness, and counts of other problems. The first two rows show a pairwise comparison between ROLLING and RE^3 and the last two rows show the equivalent comparison between ROLLING-FT and RE^3. Bolding indicates significant differences with $p < 0.05$ on a paired t-test. Workers judged stories from RE^3 as significantly more coherent and relevant to the initial premise, in addition to having fewer writing problems. 58
4.2 A heavily abridged RE^3 example generation (see [168] Appendix J, Tables 39 and 40 for complete setup and story). While certainly not perfect, the story demonstrates strong long-range coherence and faithfulness to the premise. . . . 60
4.3 Another heavily abridged RE^3 example generation (see [168] Appendix J, Tables 24 and 25 for complete setup and story). RE^3 initially fails to follow the premise and outline, and in the beginning Jaxon is incorrectly introduced as Hannah's love interest. However, both issues are corrected in the subsequent story. 61
4.4 Ablations on individual components of RE^3, removing the Plan, Rewrite, and Edit modules respectively, examining comparisons of interestingness, coherence, relevance, humanlikeness, and counts of other problems. Each two rows show a pairwise comparison experiment between RE^3 and the corresponding ablation. Bolding indicates significant differences with $p < 0.05$. Both the Plan and Rewrite module are critical to performance, but the Edit module makes little difference. 62
4.5 ROC-AUC score of predicted contradiction probabilities for different methods on our evaluation set. STRUCTURED-DETECT outperforms our two entailment-based baselines. 63

4.6 Pairwise comparisons of DOC against baselines on passages corresponding to top-level outline items from 20 stories. Bold indicates significance with $p < 0.05$. DOC stories are rated substantially more plot-coherent, outline-relevant, and interesting compared to RE^3 and ROLLING-OPT. 74
4.7 Example of a premise and heavily abridged DOC outline (settings, characters, and depth-3 items omitted; see [166] Appendix M, Table 28 for complete plan). . . 75
4.8 A heavily abridged DOC story generated from the outline shown in Table 4.7 (see [166] Appendix M, Table 29 for complete story). Although some issues remain, the story has a coherent overarching plot which follows the outline. 76
4.9 Pairwise comparison of DOC vs. RE^3 on 20 human-interactive story generation runs. Humans judged faithfulness to authorial intent, control over generation, system intuitiveness, and story quality. Numbers indicate the percentage of responses in favor of each system, with "no preference" responses omitted. Bolding indicates significance with $p < 0.05$. DOC is preferred by a wide margin on all metrics. 76
4.10 Pairwise comparisons of DOC vs. ablations without the detailed outliner and detailed controller, respectively, on passages from 10 stories. Bold indicates significance with $p < 0.05$. Although the results on plot-coherence and interestingness are inconclusive, both the detailed outliner and detailed controller are important for outline relevance. 77
4.11 Percentage of short passages that are faithful to corresponding outline leaf nodes, ablating the detailed controller. Bold indicates significance with $p < 0.05$. The detailed controller greatly improves relevance to leaf nodes. 78

Acknowledgments

I would like to thank my advisor Dan Klein, the Berkeley NLP Group, and all of my other paper coauthors for their invaluable help and advice. I would also like to thank all of my family and friends for their unwavering support throughout my PhD.

Chapter 1

Introduction

1.1 Background

Recent developments in large language models (LLMs) have substantially advanced the state of the art across a wide variety of natural language tasks [30, 15, 106]. However, while such models can generate fluent text, it may be difficult to adequately control their behavior at inference time. For example, pretrained language models out of the box produce biased or toxic text with alarming frequency [15, 89], perhaps due to the presence of such text in large quantities in their pretraining data.

Therefore, controlled generation—controlling a model's behavior at inference time to produce outputs in line with desired soft or hard constraints—is necessary to ensure ethical use of these powerful technologies in real-world settings. In fact, detoxifying problematic biases is just one of many applications which fall under the broad umbrella of controlled generation. Such applications of controlled generation range from maintaining a desired style or level of formality, to maintaining faithfulness to a reference document or to real-world factuality, or even to non-language tasks (using models for non-language domains) such as designing molecules with desirable pharmaceutical properties.

Accordingly, substantial effort has been devoted in recent years to developing methods for controlling language model outputs. Such efforts include a variety of methods which operate solely at inference time [73, 29, 164], as well as methods which rely on modifying the underlying model distribution whether by finetuning [141, 135] or reinforcement learning [6, 7].

Nevertheless, the task of controlling LLMs is highly complex, and only becomes more difficult as time passes: language models continue to improve in capability while becoming increasingly un-interpretable, we have more limited access to the strongest models which are increasingly hidden behind private APIs, and the control objectives we ask for become increasingly difficult. To expound upon the latter point: just two or three years ago we might have been content with simply controlling the general topic of an output passage just a few dozen tokens long, but today a major challenge is to control for factual accuracy with respect to both the provided context as well as to real-world knowledge, on output passages which may span thousands or even tens of thousands of tokens.

1.2 Contributions

In this thesis, I will discuss several methods I have developed that tackle controlled generation in many different settings.

Control For Natural Language

First, in Chapter 2 I will discuss general control approaches for unstructured natural language, including both purely inference-time control as well as reinforcement learning-based finetuning.

For pure inference-time control, I propose Future Discriminators for Generation (FUDGE), a flexible and modular method for controlled text generation. Given a pre-existing base LM for generating text from a distribution of interest, FUDGE enables conditioning on a desired attribute a (for example, formality) while requiring access only to the base LM's output logits. FUDGE learns an attribute predictor operating on a partial sequence, and uses this predictor's outputs to adjust the base LM's original probabilities. We show that FUDGE models terms corresponding to a Bayesian decomposition of the conditional distribution of the base LM given attribute a. Moreover, FUDGE can easily compose predictors for multiple desired attributes. We evaluate FUDGE on three tasks — couplet completion in poetry, topic control in language generation, and formality change in machine translation — and observe gains in all three tasks.

Compared to pure inference-time control, finetuning (including RL-based approaches) requires and takes advantage of greater access to the base language model. While imposing an additional upfront cost to train or finetune the model, they can reduce the cost at inference time by enabling better control through vanilla sampling from the finetuned model (compared to inference-time control approaches, which may require more expensive decoding procedures to enable control).

RL-based approaches to finetuning have become increasingly popular of late. In this thesis, I will propose Reinforcement Learning from Contrastive Distillation (RLCD), an RL method for aligning language models to follow principles expressed in natural language (e.g., to be more harmless) without using human feedback. RLCD creates preference pairs from two contrasting model outputs, one using a positive prompt designed to encourage following the given principles, and one using a negative prompt designed to encourage violating them. Using two different prompts causes model outputs to be more differentiated on average, resulting in cleaner preference labels in the absence of human annotations. We then use the preference pairs to train a preference model, which is in turn used to improve a base unaligned language model via reinforcement learning. Empirically, RLCD outperforms RLAIF [6] and context distillation [59] baselines across three diverse alignment tasks—harmlessness, helpfulness, and story outline generation—and when using both 7B and 30B model scales for simulating preference data.

Control For Structured Domains

Next, in Chapter 3, I will discuss several of my works for extending control to structured domains. Much of this section will focus on one of my works developing a method called Stochastic Iterative Target Augmentation [167] which we apply to such diverse domains as molecule design and program synthesis.

To provide some background on the molecule design task, for instance: generative models in molecular design tend to be richly parameterized, data-hungry neural models, as they must create complex structured objects as outputs. Estimating such models from data may be challenging due to the lack of sufficient training data. With Stochastic Iterative Target Augmentation, we propose a surprisingly effective self-training approach for iteratively creating additional molecular targets. We first pretrain the generative model together with a simple property predictor. The property predictor is then used as a likelihood model for filtering candidate structures from the generative model. Additional targets are iteratively produced and used in the course of stochastic EM iterations to maximize the log-likelihood that the candidate structures are accepted. A simple rejection (re-weighting) sampler suffices to draw posterior samples since the generative model is already reasonable after pre-training. We demonstrate significant gains over strong baselines for both unconditional and conditional molecular design. In particular, our approach outperforms the previous state-of-the-art in conditional molecular design by over 10% in absolute gain. Finally, we show that our approach is useful in other domains as well, such as program synthesis.

At the end of this section, I will also briefly discuss some of my other projects in this area in collaboration with others, touching on controlled generation methods for additional structured domains such as semantic parsing and crosswords.

Long-Form Story Generation

Finally, in Chapter 4, I will apply controllable generation ideas such as FUDGE [164] to the task of generating high-quality language model outputs of much longer length—up to several thousands of words—in the domain of story generation.

First, in my work on Recursive Reprompting and Revision (Re^3), we consider the problem of automatically generating longer stories of over two thousand words. Compared to prior work on shorter stories, long-range plot coherence and relevance are more central challenges here. We propose the Recursive Reprompting and Revision framework to address these challenges by (a) prompting a general-purpose language model to construct a structured overarching plan, and (b) generating story passages by repeatedly injecting contextual information from both the plan and current story state into a language model prompt. We then revise by (c) reranking different continuations for plot coherence and premise relevance, and finally (d) editing the best continuation for factual consistency. Compared to similar-length stories generated directly from the same base model, human evaluators judged substantially more of Re^3's stories as having a coherent overarching plot (by 14% absolute increase), and relevant to the given initial premise (by 20%).

We then improve further upon our previous work in Re^3 by proposing the Detailed Outline Control (DOC) framework for improving long-range plot coherence when automatically generating several-thousand-word-long stories. DOC consists of two complementary components: a detailed outliner and a detailed controller. The detailed outliner creates a more detailed, hierarchically structured outline, shifting creative burden from the main drafting procedure to the planning stage. The detailed controller ensures the more detailed outline is still respected during generation by controlling story passages to align with outline details. In human evaluations of automatically generated stories, DOC substantially outperforms Re^3 on plot coherence (22.5% absolute gain), outline relevance (28.2%), and interestingness (20.7%). Humans also judged DOC to be much more controllable in an interactive generation setting.

Lastly, I will discuss several recent and ongoing works which further explore different aspects of long-form story generation, such as personalization, pacing, and factual consistency, as well as some improvements to computational efficiency using more recent LLMs.

Chapter 2

Control For Natural Language

This chapter will mainly focus on FUDGE [164], a token-level inference-time control method, and RLCD [169], an RL-based fine-tuning alignment method. At the end I will also touch on other control methods which I have collaborated on in developing.

2.1 Future Discriminators for Generation (FUDGE)

Introduction

We can model inference-time control as conditioning on an additional desirable attribute a which is not already built into distribution $P(X)$ over text X modeled by the base LM. That is, we would like to model $P(X|a)$. For example, $P(X)$ may be a pretrained translation model for Spanish inputs I to English outputs X, but we may wish to additionally constrain the outputs to possess a new attribute a, e.g., formality, which we did not optimize for during training.

Unfortunately, once we have already obtained an unconditioned $P(X)$ defined as the output distribution of some LM, it is nontrivial to add conditioning on a new attribute a without either training a new model from scratch or fine-tuning with additional data. Although in principle we can trivially sample from $P(X|a)$ via rejection sampling from $P(X)$, rejection sampling may be highly inefficient in practice. On the other hand, while generating according to attribute a, $P(X)$ should be left otherwise intact: in the previous translation formality example, it is pointless to generate formal English outputs if they do not preserve the original Spanish meaning.

In light of these concerns, we propose FUDGE, a flexible and modular method for modeling $P(X|a)$ which accesses only the output probabilities of the generative model which defines $P(X)$. FUDGE learns a binary predictor for whether attribute a will become true in the complete future, based on an incomplete sequence prefix. Multiplying the output probabilities of this predictor with the language model's original probabilities and then renormalizing yields a model for the desired $P(X|a)$ via Bayes' Rule.

We run experiments on three controlled text generation tasks — couplet completion in poetry, topic control in language generation, and formality change in machine translation — showing our method's broad applicability. Additionally, we demonstrate the modularity of FUDGE by composing multiple attribute constraints in both the couplet and topic control tasks. In our experiments, we find that FUDGE is highly effective at attribute control, outperforming both a simple fine-tuning baseline and also a strong gradient-based method, PPLM [29].

Related Work

Ideally, a controlled text generation method should efficiently control for a while preserving $P(X)$ as much as possible. Recent work on controlled text generation has greatly advanced our ability to control for a required attribute a flexibly and cheaply, with varying degrees of modification to the original model $\mathcal{G}$ which defines $P(X)$.

One line of work fine-tunes a pretrained model for a desired attribute [39, 173, 183]. The result is a class-conditional language model (CCLM). However, it is difficult to isolate the desired attribute from the distribution shift between $\mathcal{G}$ and the fine-tuning dataset [58, 65, 80], i.e., it is nontrivial to preserve the desirable qualities of the $P(X)$ modeled by $\mathcal{G}$. One may also need to fine-tune separately for each attribute of interest. CTRL [69] partially addresses these issues by providing 55 attribute control codes for a large language model trained from scratch, although this is expensive. Very recently, GEDI [73] achieves strong performance by using CCLM generators as discriminators, though it relies on several heuristics. More broadly, text generation models for style transfer [58, 75, 28], summarization [130, 40, 175], and machine translation [76, 102, 85] can also be viewed as CCLM's for different "attributes."

A second type of approach instead conditions on a desired attribute by backpropagating gradients, either to directly modify model activations [29, 94] or to find a trigger string [149, 148]. Such methods often exhibit a high degree of attribute control, and can be used in adversarial attacks [148]. In fact, [139] show that by carefully modifying the latent state, one can cause the base $\mathcal{G}$ to produce arbitrary outputs.

A third class of methods, referred to as weighted decoding (WD), assumes access only to $P(X)$ (i.e., $\mathcal{G}$'s output logits), and operates directly on these logits [42, 55, 27, 134]. Compared to other approaches, WD methods are relatively interpretable in how they obtain $P(X|a)$ from $P(X)$, but prior WD implementations have been observed to perform poorly in controlled text generation [131, 29]. While FUDGE shares a Bayesian motivation with other WD methods, FUDGE follows the Bayesian factorization more closely in implementation. The key distinguishing feature of FUDGE is that it models whether attribute a will be true in the *future*, rather than in the *present*. We find that FUDGE substantially outperforms previous WD approaches in our experiments.

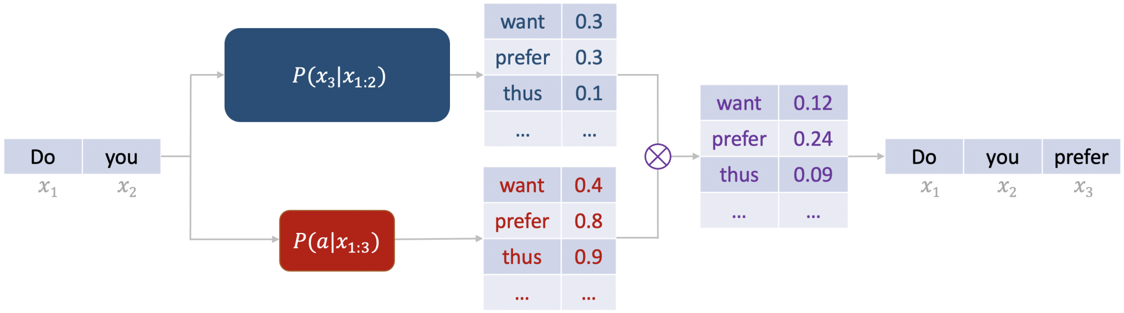


Figure 2.1: Illustration of one decoding step in FUDGE, for an example where the desired attribute a is formality. A large pretrained model $\mathcal{G}$ (dark blue) outputs unconditioned probabilities. Our binary predictor (red) predicts whether the eventual completed sequence will be formal for each possible continuation (computed for each candidate x_3, e.g., "want"; holding a fixed). The probabilities for each x_3 are multiplied (purple) and then renormalized to obtain $P(x_3|x_{1:2}, a)$, from which we sample the next token $x_3 =$"prefer."

Future Discriminators for Generation

We now explain the details of our proposed method, Future Discriminators for Generation (FUDGE), and show that it corresponds to modeling the desired conditional distribution $P(X|a)$.

For a given language generation task, assume we have an autoregressive model $\mathcal{G}$ (e.g., a large pretrained language model) which models $P(x_i|x_{1:i-1})$ for tokens $x_1 \ldots x_i$. Letting $X = x_{1:n}$ denote a completed sequence, $\mathcal{G}$ can sample from $P(X) = P(x_{1:n})$ one token at a time by factoring $P(X)$:

$$P(X) = \prod_{i=1}^{n} P(x_i|x_{1:i-1})$$

To condition on attribute a, we instead model $P(X|a)$. This requires a model for $P(x_i|x_{1:i-1}, a)$, modifying the previous factorization:

$$P(X|a) = \prod_{i=1}^{n} P(x_i|x_{1:i-1}, a)$$

If we model $P(x_i|x_{1:i-1}, a)$ directly, we obtain a class-conditional language model (CCLM). We can learn the CCLM by e.g., fine-tuning $\mathcal{G}$ depending on the available data, possibly with some structural modification to $\mathcal{G}$ to accommodate conditioning.

However, FUDGE instead relies on the following Bayesian factorization, exchanging x_i and a conditioned on $x_{1:i-1}$:

$$P(x_i|x_{1:i-1}, a) \propto P(a|x_{1:i})P(x_i|x_{1:i-1})$$

The second term is exactly the quantity modeled by the base $\mathcal{G}$. It then suffices to model the first term, $P(a|x_{1:i})$, with a binary classifier $\mathcal{B}$ for the attribute a given a prefix $x_{1:i}$. Intuitively, one can view $\mathcal{B}$ as rescoring or reranking $\mathcal{G}$'s original hypotheses.

We emphasize that although $\mathcal{B}$ takes a *prefix* $x_{1:i}$ as input, it predicts whether attribute a will *in the future* be satisfied for the *completed* generation $x_{1:n}$. For instance, suppose we are given a dataset of examples $\{(x_{1:n}, a')\}$ with a' being the values of binary indicators for the desired a (i.e., if a is formality, then a' is 0 or 1 when $x_{1:n}$ is informal or formal respectively). For each training example $(x_{1:n}, a')$, we train our classifier $\mathcal{B}$ using all pairs $(x_{1:i}, a')$; that is, we construct a separate example from each prefix $x_{1:i}$ of $x_{1:n}$. Our approach contrasts with previous methods such as [29], which greedily optimize for a on the immediate extension $x_{1:i+1}$. One particular benefit is that FUDGE naturally plans for the future: in the example for generating text on the "space" topic in Table 2.6, FUDGE writes about a "mysterious ship" despite "ship" itself not being in the given "space"-topic bag of words, because "mysterious ship" easily leads into a mention of one of the targeted "space" words ("Earth"). Similarly, in the first couplet completion example in Table 2.3, FUDGE needs to rhyme with "fear" after exactly ten syllables. After seven syllables, it could reasonably generate the word "clear," but it first generates the adverb "pretty" in order to set up the generation of "clear" as the tenth syllable.

FUDGE's implementation is shown schematically in Figure 2.1, and is quite simple in practice. FUDGE just needs to learn a $\mathcal{B}$ (red in Figure 2.1) sharing tokenization with $\mathcal{G}$ (dark blue). It then converts $\mathcal{B}$'s output into probabilities (red table in Figure 2.1), and multiplies with the original output probabilities from $\mathcal{G}$ (dark blue table), to obtain unnormalized probabilities $P(x_i, a|x_{1:i-1})$ (purple table). Finally, renormalizing over the output vocabulary yields the desired distribution $P(x_i|x_{1:i-1}, a)$. In practice, we operate in the log-probability space for numerical stability.

To improve computational efficiency, we typically choose $\mathcal{B}$ to be lightweight relative to $\mathcal{G}$. We also consider only the top 200 possibilities for x_i according to $\mathcal{G}$ at each step, as a cheap approximation to the full distribution, and find that this works well in practice.[1] In each task in our experiments, running FUDGE on the test set takes no more than 15 minutes on a single Quadro RTX 6000 GPU.

Finally, as with other controlled generation approaches such as [29], it is likely that augmenting FUDGE with reranking approaches such as rejection sampling could improve output quality at the cost of compute time, although we do not comprehensively evaluate such extensions in this work.

Advantages and Limitations

We highlight several additional potential advantages of FUDGE compared to directly modeling $P(x_i|x_{1:i-1}, a)$ via e.g., a fine-tuned CCLM:

1. FUDGE requires access only to $P(X)$ (i.e., $\mathcal{G}$'s output logits) rather than $\mathcal{G}$ itself.

---

[1] See [164] Appendix H for ablations on the top-200 pruning.

2. $\mathcal{G}$ can be freely swapped out for any other model that shares the same tokenization when larger models become available.

3. Given multiple conditionally independent attributes with predictors for each, FUDGE can easily condition on the combination of these attributes in a modular fashion by summing their output log-probabilities.

Unfortunately, like previous methods, FUDGE cannot fully guarantee that all outputs possess the desired attribute a. In FUDGE's case, this is due to the approximation inherent in modeling $P(a|x_{1:i})$, as well as only considering the top 200 possible x_i for computational efficiency.

Experiments

We run experiments on a range of controlled text generation tasks to evaluate the effectiveness of our proposed method: poetry couplet completion, topic-controlled language generation, and machine translation formality change. For each task we discuss the evaluation setup, the specific details of our method and baselines, and finally experimental results.

Poetry Couplet Completion

So long as men can breathe or eyes can see, So long lives this and this gives life to thee.

Table 2.1: An example couplet by William Shakespeare. Every second syllable is stressed, following iambic meter, and the last words of each line (see/thee) rhyme.

We begin with English poetry generation, a task that emphasizes well-formedness, and which has been studied in different forms by many previous works [178, 151, 41, 42]. Our task here is couplet completion. Given the first line of an iambic pentameter couplet (e.g., Table 2.1), the model must generate a second line which (1) satisfies iambic pentameter, (2) rhymes with the first line, and (3) ends a sentence. The desired attribute a is defined as possessing all three properties, as evaluated by a rule-based checker $\mathcal{F}$ ([164] Appendix A). Our test set is a collection of prefix lines of couplets, collected from the ending couplet of each of Shakespeare's 154 sonnets.

Metrics. We consider four metrics.

1. *Success*, the fraction of couplet completions with the desired attribute a, as checked by $\mathcal{F}$. This is the main metric.

2. *Grammaticality*, the probability of grammaticality given by a Roberta-based CoLA grammaticality model [95, 156], averaged over all outputs.

3. *Perplexity* of the completion conditioned on the prefix. Following [29], since our models use GPT2-Medium [117] as $\mathcal{G}$, we evaluate perplexity using GPT [118].[2]

4. *Distinctness* of completions, measured as the number of unique unigrams, bigrams, and trigrams across all samples, divided by the total number of words [88].

At test time, we decode until the model generates ten syllables followed by an end-of-sentence punctuation mark, or after the eleventh syllable (an automatic failure, since iambic pentameter requires exactly ten syllables).

Overall, because we define a using a rule-based $\mathcal{F}$ which is accessible during training, our formulation of couplet completion is a relatively clean task for evaluating the effectiveness of FUDGE.

Fudge Instantiation. The obvious approach is to learn a predictor for $\mathcal{F}$ directly. However, the three components of a — meter, rhyme, and sentence-ending — should be roughly independent. Thus we assume conditional independence, and demonstrate the modularity of FUDGE by constructing three separate predictors to be combined at test time:

1. $\mathcal{B}_1(x_{1:i})$ takes a text prefix $x_{1:i}$, and predicts whether the completion $x_{1:n}$ of prefix $x_{1:i}$ will be in iambic meter. The model is an LSTM followed by a linear output layer.

2. $\mathcal{B}_2(x_{1:i}, t, r)$ takes prefix $x_{1:i}$, the number of syllables t between x_i and x_n for $n \geq i$, and a rhyme sound r.[3] It predicts whether the completion $x_{1:n}$ has the rhyme sound r at the

	Correctness	***Text Quality***		***Diversity***		
Method	**Success ↑**	Grammar ↑	Perplexity ↓	Dist-1 ↑	Dist-2 ↑	Dist-3 ↑
$\mathcal{G}$	0	0.52	44.3 ± 42.2	0.35	0.74	0.77
FINETUNE	0.21	0.44	55.8 ± 98.3	0.35	0.74	0.78
PPLM	0	0.54	60.8 ± 66.1	0.40	0.78	0.78
FUDGE	**0.44**	0.44	70.9 ± 89.4	0.40	0.79	0.78
Shakespeare	**0.45**	0.29	333.8 ± 418.9	0.44	0.81	0.79

Table 2.2: Couplet completion results. Success (main metric), grammaticality, perplexity, and distinctness of different methods, tested on 154 prefix lines from Shakespeare sonnets. FUDGE substantially outperforms automated baselines on success and maintains high diversity, although quality unsurprisingly suffers compared to the base $\mathcal{G}$ due to the difficult constraint $\mathcal{F}$. Note Shakespeare's work is often "incorrect" due to the narrowness of our metric $\mathcal{F}$; he also scores poorly on text quality because our evaluation models are intended for more modern English.

---

[2] See [164] Appendix E for other perplexity measurements.

[3] Two words have the same "rhyme sound" r if they rhyme according to the CMU Pronouncing Dictionary [157].

end of token x_n. The model is an LSTM with attention dependent on t and r, followed by a shallow feedforward network, and is trained via noise-contrastive estimation [51].[4]

3. $\mathcal{B}_3(x_{1:i}, t)$ takes prefix $x_{1:i}$ and the number of syllables t between x_i and x_n for $n \geq i$, and predicts whether x_n ends a sentence. The model is an LSTM followed by a shallow feedforward network.

The predictors vary in architecture because $\mathcal{B}_2$ and $\mathcal{B}_3$ require inputs other than $x_{1:i}$ — in truth, they are *families* of related predictors. We find that performance is not overly sensitive to the particulars of the predictor architectures ([164] Appendix D).

To train the discriminators, we sample a dataset of 10 million generations of varied length from GPT2-Medium. From these generations, we sample random subsequences $x_{1:n}$ of roughly 10 to 30 syllables and truncate $t \leq 10$ ending syllables. These truncations become inputs $x_{1:i}$ to the predictors. For simplicity, we did not balance the class labels for e.g., the iambic predictor during training, although it is likely that doing so would improve performance.

At test time, we extract r from the given first line of the couplet, and initialize $t = 10$, updating at each step. We then modify the output logits of $\mathcal{G}$ by simply adding the log-probabilities from $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$, demonstrating the ease of composing constraints in FUDGE.

Baselines. We compare to four baselines.[5]

1. $\mathcal{G}$, the original GPT2-Medium.

2. FINETUNE, a CCLM which finetunes $\mathcal{G}$ on similar inputs to those used for $\mathcal{B}_2$ in FUDGE. Since it is not obvious how to compose multiple CCLM's for different attributes, we train a single CCLM for all desired properties together. We condition by prefixing the input with (1) whether the last 10 syllables of the original untruncated $x_{1:n}$ are iambic, (2) the rhyme sound at the end of x_n, and (3) whether a sentence ends with x_n. A special token is inserted 10 syllables from the end of $x_{1:n}$.

3. PPLM [29], which uses shallow predictors learned from $\mathcal{G}$'s top-level hidden layer to modify $\mathcal{G}$'s states toward increasing probability of the desired attribute via gradient ascent. We decompose the predictors into the same iambic, rhyme sound, and end-of-sentence predictors as for FUDGE, inserting an additional hidden layer in the shallow

---

[4] The output logits from $\mathcal{B}_2$ are unnormalized, but this does not affect FUDGE after they are added to the output logits of $\mathcal{G}$ and softmaxed for sampling.

[5] A system like Hafez [41, 42], which enforces meter and rhyme at each decoding step using a hard constraint, could achieve perfect success rate. However, this approach relies on the meter and rhyme attributes being "prefix-checkable" at the word level: one can guarantee success by simply never selecting a word which immediately violates the constraint. This is often the case for simple rule-based constraints, but not for many other interesting attributes, such as the topic and formality attributes in our subsequent experiments. To preserve generality, FUDGE does not rely on this "prefix-checkable" property, and neither do our baselines.

predictor when needed to incorporate additional input (the desired rhyme sound and/or number of syllables until end-of-sentence).

4. Shakespeare's original couplet completions.

All non-Shakespeare methods use top-k sampling with $k = 10$.

Results. Even though our GPT2-Medium-generated training dataset is completely different from the test domain, and contains essentially zero examples of correct couplets, FUDGE is able to learn the desired attribute. As shown in Table 2.2, FUDGE greatly outperforms all automated baselines in success rate.

Surprisingly, the PPLM baseline achieves zero success. We find that its iambic and rhyme predictors are very poor, so we hypothesize that the relevant information is not easily extractable from the last hidden layer of $\mathcal{G}$. In contrast, FUDGE's predictors operate directly on the raw text.

Funnily enough, FUDGE even matches Shakespeare according to $\mathcal{F}$, although this is largely due to the narrowness of $\mathcal{F}$ and should not be taken seriously.[6] Similarly, the grammaticality and perplexity metrics are designed for our automated baselines, and thus assign poor scores to Shakespeare's antiquated and flowery style.

FUDGE also maintains relatively fluent generation despite lower grammaticality and perplexity compared to $\mathcal{G}$. See Table 2.3 for two successful examples. Interestingly, FUDGE also increases diversity compared to $\mathcal{G}$, perhaps due to the difficult constraint $\mathcal{F}$ forcing FUDGE to use lower-probability regions of the base distribution $P(X)$.

And even thence thou wilt be stol'n, I fear, for this shall be the end. That's pretty clear.
Or, if they sleep, thy picture in my sight I will be glad to look upon the night.

Table 2.3: Two examples of successful couplet completions (in purple) generated by FUDGE.

Finally, it is possible (and trivial) to adjust the conditioning strength in FUDGE by multiplying the binary predictors' output logits by a constant. However, this deviates from our Bayesian factorization of $P(X|a)$, and we do not do so.

---

[6] We define $\mathcal{F}$ using somewhat narrow criteria ([164] Appendix A), which capture only a subset of what Shakespeare considered to be well-written couplets. The purpose of this task is to evaluate FUDGE's ability to satisfy a difficult well-formedness constraint compared to automated baselines, rather than to perfectly capture the human notion of an iambic pentameter couplet. Thus Shakespeare is marked wrong when he (1) uses archaic pronunciations, (2) uses loose rhymes, (3) elides syllables to fit meter, or (4) uses words missing from the CMU Pronouncing Dictionary. See [164] Appendix A.1 for details. Of course, Shakespeare is only included as a whimsical point of reference; our generations obviously do not hold a candle to Shakespeare's originals.

Topic-Controlled Language Generation

Next, we explore topic control in English language generation. The desired attribute a is to be on-topic for a given topic, such as science or politics. To facilitate comparison with prior work, we largely follow the setup of Pplm [29]: the model is provided an approximation to the topic at test time, in the form of a bag of on-topic words $\mathcal{W}$. The goal is to sample text according to the topic approximated by $\mathcal{W}$, starting from a generic prefix. There are 7 topics (space, politics, military, legal, science, religion, and computers) and 20 prefixes, and the model generates 3 80-token[7] samples from each topic-prefix pair, for a total of 420 generations.

Metrics. Unfortunately, we cannot easily construct a rule-based $\mathcal{F}$ for being "on-topic." Additionally, use rate of words in $\mathcal{W}$ is a poor metric, because a model can score highly by e.g., simply returning the words in $\mathcal{W}$, without generalizing to the full topic that $\mathcal{W}$ approximates. Instead, we adopt a notion of success which requires the model to generalize the bag $\mathcal{W}$ to the full topic. The remaining metrics are measures of quality and diversity.

1. *Success*, the average number of distinct words in a heldout bag $\mathcal{W}'$ which appear in the model output. Specifically, for each word in $\mathcal{W}$, we add to $\mathcal{W}'$ the closest GloVe [113] word by cosine similarity, such that the new word does not contain (and is not contained by) any word in $\mathcal{W}$. (This excludes e.g., most plurals.) Usage of distinct words in $\mathcal{W}'$ measures the model's ability to generalize $\mathcal{W}$ to other on-topic words, of which $\mathcal{W}'$ is a non-exhaustive set. This is our main metric.
2. *Grammaticality*, identical to the couplet task.
3. *Perplexity*, identical to the couplet task.

---

[7] All models and baselines use GPT2 tokenization.

	On-Topic	***Text Quality***		***Diversity***		
Method	**Success ↑**	Grammar ↑	Perplexity ↓	Dist-1 ↑	Dist-2 ↑	Dist-3 ↑
$\mathcal{G}$	0.22	0.81	37.1 ± 26.9	0.35	0.78	0.92
Finetune	0.28	0.74	24.9 ± 13.7	0.29	0.70	0.88
Wdec	0.14	0.59	33.8 ± 33.7	0.16	0.42	0.55
Pplm	0.48	0.78	43.1 ± 23.7	0.35	0.78	0.92
Fudge	**0.59**	0.79	40.7 ± 26.3	0.34	0.75	0.91

Table 2.4: Topic control results. Success (main metric), grammaticality, perplexity, and distinctness for different methods. Finetune and Wdec often degenerate into repeating the given bag of words $\mathcal{W}$; this is ill-captured by perplexity, but results in poor grammaticality and distinctness. Fudge substantially outperforms all baselines on success, including the strong gradient-based Pplm baseline, while preserving high quality and diversity.

4. *Distinctness*, defined as in the couplet task. However, it is calculated separately within the 60 generations for each topic, and then averaged over the 7 topics.

Additionally, following the evaluation procedure of prior work such as [29], we run human evaluations via Amazon Mechanical Turk for FUDGE against each baseline, comparing topic control and fluency. For each pairwise comparison, we ask 3 workers to evaluate each of 420 paired outputs. Workers were asked to mark which generation is more on topic (first, second, both, or neither), and to rate each generation's fluency on a Likert scale from 1 to 5. We report the average fraction of outputs marked as on-topic as well as the average fluency rating for each method.

Fudge Instantiation. Since we model topics as bags of words, FUDGE uses a binary predictor $\mathcal{B}(x_{1:i}, w)$ which takes a prefix $x_{1:i}$ and word w, and classifies whether w appears in the future $x_{i:n}$ for $n \geq i$. (Since it is desirable to *stay* on topic even after successfully *getting* on topic, we use $x_{i:n}$ rather than $x_{1:n}$.) Training examples $(x_{1:i}, w)$ are sampled from the same dataset of 10 million GPT2-Medium generations used for the couplet task, and $\mathcal{B}$ is trained using noise-contrastive estimation. $\mathcal{B}$ is a lightweight LSTM-based classifier similar to $\mathcal{B}_2$ from the couplet task.

At test time, we can compose individual-word constraints if we assume conditional independence between words (although this may be imperfect). Given a bag of N words $\{w_1 \dots w_N\}$ and prefix $x_{1:i}$, we could condition on all words in the bag appearing in the future by adding all log-probabilities $\log P(w_1|x_{1:i}) \dots \log P(w_N|x_{1:i})$ to $\mathcal{G}$'s logits. However, topic control does not require every word to appear; perhaps some number λ of on-topic words is enough to be "on-topic." Therefore, we model the topic constraint as selecting a random subset of λ words from the original bag, and requiring that only those λ words all appear. Since each of the N words is selected with probability $\frac{\lambda}{N}$, the quantity we add to the base $\mathcal{G}$ logits is $\frac{\lambda}{N} \sum_{j=1}^{N} \log P(w_j|x_{1:i})$ in expectation. In our experiments we use $\lambda = 4$, based on a fantasy-topic bag of words used for validation ([164] Appendix C).

Baselines. We compare to four baselines.

1. $\mathcal{G}$, the original GPT2-Medium.

2. FINETUNE, which finetunes $\mathcal{G}$ on the same inputs used for FUDGE. The future word is given as a prefix for conditioning. At test time, we compute logits for each prefix in the given $\mathcal{W}$ and use the average as the true logits, as an ad hoc way to condition on the full $\mathcal{W}$.

3. WDEC, a simple weighted decoding implementation which greedily considers only the immediate next token when optimizing for a. Instead of using $\mathcal{B}$, WDEC just adds a fixed λ_{WDEC} to the logit for each word in $\mathcal{W}$. Note WDEC requires a to be well-defined at the token level, so it is not easily transferable to certain tasks (e.g., couplet completion).

4. PPLM [29], which modifies the activations of $\mathcal{G}$ to make the desired bag of words more likely at the immediate next position. We use their method without reranking for fair comparison.

All methods use top-k sampling with $k = 10$, following [29]'s setup.

Method	**Topic**	**Fluency**
$\mathcal{G}$	0.16	4.11
FUDGE	**0.78**	**4.30**
FINETUNE	0.24	3.95
FUDGE	**0.76**	**4.22**
WDEC	0.49	2.50
FUDGE	**0.75**	**4.21**
PPLM	0.45	4.05
FUDGE	**0.74**	**4.16**

Table 2.5: Topic control human evaluations, pairwise comparisons. FUDGE achieves a substantially higher fraction of on-topic outputs compared to each baseline, in addition to higher average fluency (rated 1 to 5).

Results. FUDGE achieves the highest success by a substantial margin (Table 2.4), and outperforms all baselines on human evaluations in both topic relevance and fluency (Table 2.5). FUDGE simultaneously preserves high quality and diversity according to automated metrics. Table 2.6 shows two examples.

Unsurprisingly, $\mathcal{G}$ performs poorly on success. WDEC and FINETUNE also perform poorly, in success and especially in distinctness. WDEC frequently degenerates into repeating the given words in the bag $\mathcal{W}$, despite tuning λ_{WDEC} ([164] Appendix C). FINETUNE also suffers from repetition, which appears to be the result of distribution shift from fine-tuning. Our fine-tuning dataset was built by sampling directly from the original $P(X)$ modeled by $\mathcal{G}$ to mitigate distribution shift, but it is well-known that language model generations are more repetitive than natural language [55, 54]. We hypothesize that FINETUNE, being fine-tuned on language model generations rather than natural language, amplifies this repetitiveness. This repetition is reflected in the poor grammaticality for both FINETUNE and especially WDEC. In contrast, FUDGE does not touch the original $P(X)$, largely avoiding FINETUNE's distribution shift problem on this task.

Finally, FUDGE outperforms the strong gradient-based PPLM method, despite requiring access only to $\mathcal{G}$'s output logits. Non-reliance on gradients means FUDGE is also many times faster than PPLM, which takes a few hours compared to FUDGE's 15 minutes for the full set of 420 generations on our hardware. Sometimes we do not even have gradients: for example, gradients are unavailable in the API for GPT3 at time of writing.

Space: The issue focused on the original plot, which was about a mysterious ship that would land on Earth, and would lead to humanity's first interstellar expedition. The original plan called for humanity to use the spacecraft to colonize outer space and build the first city on Mars. But this idea fell by the wayside in the final drafts.\n\n”It was just not a very popular idea and it wasn'
Politics: The issue focused on whether the two institutions were operating within the bounds set by the constitution and the law.\n\nThe Constitutional Court said that both governments ”have a duty to ensure the integrity of the electoral process and its effective administration, especially in light of the current political climate that is threatening the functioning of elections”

Table 2.6: The first output from FUDGE when using the prefix "The issue focused on" for two topics. We use red to highlight words in the given bag of words $\mathcal{W}$ along with obvious forms (e.g., plurals), and cyan for other on-topic words, including related words not in the heldout bag $\mathcal{W}'$. More examples in [164] Appendix J.

Machine Translation Formality Change

Finally, we turn to a somewhat more challenging task, changing formality in machine translation — specifically, from informal to formal. Given a source sentence written in an informal and conversational style, the goal is to output a translation which is also more formal. We test on the Fisher and CALLHOME Spanish–English Speech Translation Corpus [115], a collection of transcribed Spanish conversations with English translations. Both the source Spanish and target English are highly informal and disfluent. [126] augment the Fisher dataset with additional parallel English translations, rewritten to be more fluent (and hence more formal); see Table 2.7 for an example. Our task is to translate the original informal Spanish to into more formal English. However, we assume that [126]'s fluent references are unavailable during training.

entonces de verdad sí sí pero entonces tu estudiando para es es digo es más porque es exactamente
Then, if it's business, but then you are a student for a PHD, the Master's is that exactly.
If it's business, then you are a student for a PhD. The masters is exactly that.

Table 2.7: An example from the Fisher dataset.
Top: The original Spanish transcription.
Middle: The original English translation.
Bottom: [126]'s more fluent version.

Metrics. The desired attribute a is formality, but we cannot sacrifice the source sentence's meaning. The latter requirement makes generation more constrained than in the couplet and topic tasks, so perplexity and distinctness are less relevant. Instead, we use the following:

1. *BLEU Score* [109], using two of [126]'s fluent references per test example. This is our main metric.
2. *Formality*, the average probability that the model's outputs are formal, according to an evaluator trained on the Family/Relationships domain of the GYAFC formality dataset [122]. The evaluator is an LSTM followed by a linear layer.

Fudge Instantiation. We assume that the attribute a, formality, is conditionally independent from the original conditioning in $\mathcal{G}$, i.e., the meaning of the Spanish input. FUDGE uses a binary predictor $\mathcal{B}(x_{1:n})$ which classifies whether the text starting with prefix $x_{1:n}$ is written in a formal style. $\mathcal{B}$ is an LSTM followed by a linear layer, trained on the Entertainment/Music domain of GYAFC.

At test time, FUDGE directly augments $\mathcal{G}$'s logits using log-probabilities from $\mathcal{B}$. $\mathcal{G}$ is a pretrained Marian [66] transformer model for Spanish-English. We evaluate both when $\mathcal{G}$ is fine-tuned on the original Fisher training dataset (i.e., using the original targets, not [126]'s more fluent targets) as well as zero-shot with no fine-tuning, which is challenging due to the highly informal and disfluent text.

Baselines. We compare to two baselines.

1. $\mathcal{G}$, the original machine translation model.
2. $\mathcal{G}$ + ST, a pipeline consisting of $\mathcal{G}$ followed by a style transfer model. Our style transfer model is T5 [120], fine-tuned on the same GYAFC Entertainment/Music domain that we used to train $\mathcal{B}$ in FUDGE.

Since we do not assume access to [126]'s more formal targets during training, it is difficult to apply PPLM to this task: PPLM's predictor would operate on the pretrained translation model's hidden states, thus requiring a Spanish-English translation dataset with both formal and informal English.[8] We omit FINETUNE for the same reason. In contrast, FUDGE requires only the original English dataset with formality annotations.

All methods use greedy decoding.

Results. As shown in Table 2.8, FUDGE increases the formality of outputs compared to $\mathcal{G}$, even though the test-time formality predictor is trained on a different domain (Family/Relationships, rather than Entertainment/Music). Note that formality unsurprisingly decreases after fine-tuning $\mathcal{G}$, simply due to the informality of the fine-tuning dataset. As in the couplet task, one could adjust the strength of the formality control in FUDGE, although this is unprincipled from the view of modeling $P(X|a)$.

---

[8] We nevertheless ran PPLM in a somewhat convoluted setup, but found that it performed poorly ([164] Appendix B).

	$\mathcal{G}$ *(No fine-tune)*		$\mathcal{G}$ *(Fine-tune)*	
Method	**BLEU** ↑	Form. ↑	**BLEU** ↑	Form. ↑
$\mathcal{G}$	16.98	0.45	**22.03**	0.41
$\mathcal{G}$ + ST	7.87	0.96	9.63	0.97
FUDGE	**17.96**	0.51	**22.18**	0.48

Table 2.8: Machine translation formality results. BLEU (main metric) and average formality for different methods, with and without fine-tuning $\mathcal{G}$ on the Fisher domain. FUDGE increases the formality of translations compared to the base model $\mathcal{G}$ while preserving or increasing BLEU score. Conversely, $\mathcal{G}$ with style transfer overfits to the formality data, resulting in near-perfect formality but losing the original meaning.

Moreover, while FUDGE and $\mathcal{G}$ achieve similar BLEU after fine-tuning $\mathcal{G}$, FUDGE achieves higher BLEU compared to $\mathcal{G}$ when $\mathcal{G}$ is not fine-tuned on the Fisher training set. In the latter case, controlling for formality somewhat remedies the struggles of $\mathcal{G}$ when not fine-tuned on such disfluent text.

In contrast, the $\mathcal{G}$ + ST baseline achieves near-perfect formality but less than half the BLEU of $\mathcal{G}$, due to the style transfer model overfitting to the GYAFC Entertainment/Music dataset. This is similar to the distribution shift issue that we observed in topic control for FINETUNE, an issue which FUDGE largely avoids. Nevertheless, there remains substantial room for improvement on this difficult task.

Spanish	que era lo que tenía que tienes que hacer
$\mathcal{G}$	that was what you had to do
FUDGE	That was what you had to do
Reference	What's there to do?
Spanish	ah en mi en inglaterra por ejemplo
$\mathcal{G}$	Ah, in my, in England, for example.
FUDGE	Ah, in England, for example.
Reference	In England, for example?

Table 2.9: Example translations by $\mathcal{G}$ (fine-tuned on the Fisher dataset) and FUDGE using the same $\mathcal{G}$. Original Spanish and [126] references also shown. In this setting, FUDGE achieves similar BLEU to $\mathcal{G}$ while increasing formality. While FUDGE often simply corrects punctuation or capitalization (top), it also makes more complex adjustments (bottom). More examples in [164] Appendix L.

Discussion

FUDGE is a principled approach to controlled text generation which models $P(X|a)$ by closely following a Bayesian factorization, thus preserving the base $P(X)$ as much as possible. FUDGE achieves strong performance on a wide range of different tasks: poetry couplet completion, topic control, and informal-to-formal machine translation. Additionally, FUDGE can easily compose different attributes in a modular fashion: the meter, rhyme, and end-of-sentence constraints for couplet completion, and the individual words within each topic bag for topic control. In principle, FUDGE is applicable to any controlled generation task where we can train discriminators for the desired attribute or attributes.

2.2 Reinforcement Learning from Contrastive Distillation (RLCD)

We now shift our focus from pure inference-time control to a finetuning-based alignment method.

Introduction

Reinforcement Learning from Human Feedback (RLHF) has recently been used to great effect to align pretrained large language models (LLMs) to human preferences, optimizing for desirable qualities like harmlessness and helpfulness [7] and achieving state-of-the-art results across a variety of natural language tasks [106].

A standard RLHF procedure fine-tunes an initial unaligned LLM using an RL algorithm such as PPO [128], optimizing the LLM to align with human preferences. RLHF is thus critically dependent on a reward model derived from human-labeled preferences, typically *pairwise preferences* on LLM outputs (o_1, o_2) generated from a shared prompt p.

However, collecting human pairwise preference data, especially high-quality data, may be expensive and time consuming at scale. To address this problem, approaches have been proposed to obtain labels without human annotation, such as Reinforcement Learning from AI Feedback (RLAIF) and context distillation.

RLAIF approaches (e.g., [6]) simulate human pairwise preferences by scoring o_1 and o_2 with an LLM (Figure 2.2 center); the scoring LLM is often the same as the one used to generate the original pairs (o_1, o_2). Of course, the resulting LLM pairwise preferences will be somewhat noisier compared to human labels. However, this problem is exacerbated by using the same prompt p to generate both o_1 and o_2, causing o_1 and o_2 to often be of very similar quality and thus hard to differentiate (e.g., Table 2.10). Consequently, training signal can be overwhelmed by label noise, yielding lower-quality preference data.

Meanwhile, context distillation methods (e.g., [141]) create more training signal by modifying the initial prompt p. The modified prompt p_+ typically contains additional context encouraging a *directional attribute change* in the output o_+ (Figure 2.2 right). However, con-

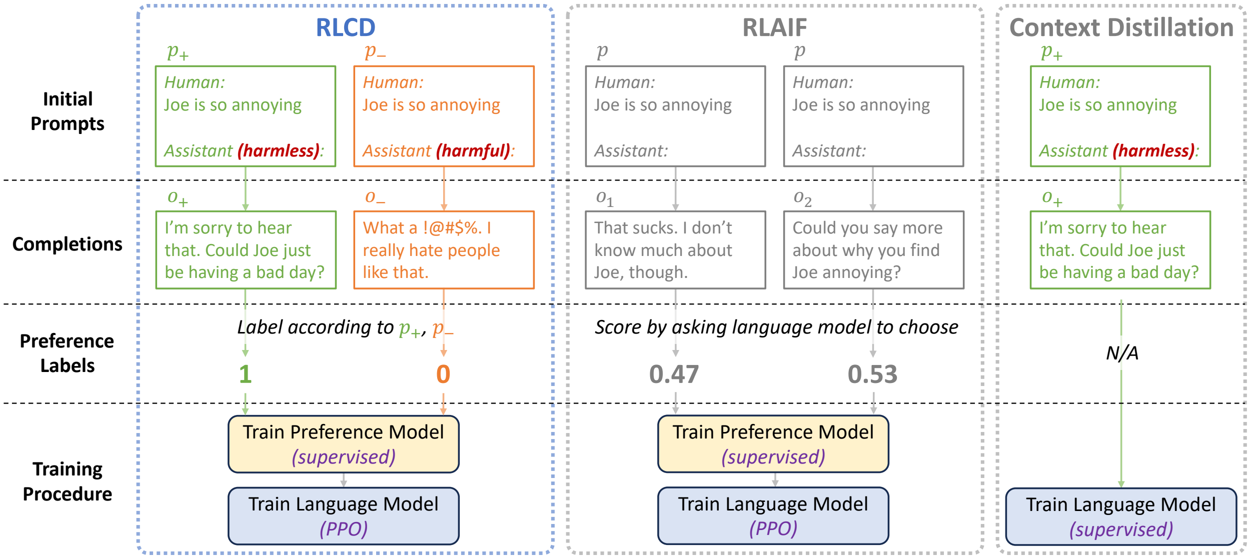

RLCD
RLAIF
Context Distillation
Initial Prompts
Completions
Preference Labels
Training Procedure
p_+
p_-
p
Human: Joe is so annoying Assistant (harmless):
Human: Joe is so annoying Assistant (harmful):
Human: Joe is so annoying Assistant:
o_+
o_-
o_1
o_2
I'm sorry to hear that. Could Joe just be having a bad day?
What a !@#$%. I really hate people like that.
That sucks. I don't know much about Joe, though.
Could you say more about why you find Joe annoying?
Label according to p_+, p_-
Score by asking language model to choose
N/A
1
0
0.47
0.53
Train Preference Model (supervised)
Train Language Model (PPO)
Train Language Model (supervised)

Figure 2.2: Stylized illustration showing RLCD compared to standard RLAIF and context distillation on harmlessness attribute. RLCD generates preference pairs using two contrasting prompts p_+, p_-, and labels according to the prompt used, thus making use of both pairwise preferences for RL as well as directional attribute change in outputs as encouraged by prompts. RLCD then trains a preference model on the resulting pairs, which is used to guide the LLM alignment via PPO.

text distillation methods only generate a single output o_+ per prompt p_+, which is then used for supervised fine-tuning, losing the pairwise preferences which help RLHF-style approaches to derive signal from the contrast between outputs. Multiple works have observed that RL approaches using preference models for pairwise preferences can substantially improve over supervised fine-tuning by itself when aligning LLMs [108, 32].

Therefore, while both RLAIF and context distillation approaches have already been successfully applied in practice to align language models, we posit that it may be even more effective to combine the key advantages of both. That is, we will use RL with *pairwise preferences*, while also using modified prompts to encourage *directional attribute change* in outputs.

Concretely, we propose Reinforcement Learning from Contrastive Distillation (RLCD). RLCD generates preference data as follows. Rather than producing two i.i.d. model outputs (o_1, o_2) from the same prompt p as in RLAIF, RLCD creates two variations of p: a *positive prompt* p_+ similar to context distillation which encourages directional change toward a desired attribute, and a *negative prompt* p_- which encourages directional change *against* it (Figure 2.2 left). We then generate model outputs (o_+, o_-) respectively, and automatically label o_+ as preferred—that is, RLCD automatically "generates" pairwise preference labels by construction. We then follow the standard RL pipeline of training a preference model followed by PPO.

Compared to RLAIF-generated preference pairs (o_1, o_2) from the same input prompt p, there is typically a clearer difference in the quality of o_+ and o_- generated using RLCD's directional prompts p_+ and p_-, which may result in less label noise. That is, intuitively, RLCD exchanges having examples be *closer to the classification boundary* for *more accurate labels* on average. Compared to standard context distillation methods, on top of leveraging pairwise preferences for RL training, RLCD can derive signal not only from the positive prompt p_+ which improves output quality, but also from the negative prompt p_- which degrades it. Positive outputs o_+ don't need to be perfect; they only need to contrast with o_- on the desired attribute while otherwise following a similar style.

We evaluate the practical effectiveness of RLCD through both human and automatic evaluations on three tasks, aiming to improve the ability of LLaMA-7B [145] to generate harmless outputs, helpful outputs, and high-quality story outlines. RLCD substantially outperforms both RLAIF and context distillation baselines in pairwise comparisons when simulating preference data with LLaMA-7B, while still performing equal or better when simulating with LLaMA-30B.

Related Work

Lately, several RL approaches leveraging reward models trained on human preferences [108, 7, 181, 119] have been applied to align strong pretrained LLMs [137, 107, 106, 4, 144]. However, it can be expensive to collect human pairwise preferences.

Reinforcement Learning from AI Feedback. RLAIF simulates human pairwise preferences using a LLM, whether the same LLM to be aligned later [6] or a stronger LLM as an oracle [32]. Such methods typically obtain pairwise preferences by scoring two i.i.d. outputs (o_1, o_2). RLCD instead generates outputs (o_+, o_-) from different distributions, obviating the need for post hoc scoring (whether human or AI).

Context Distillation. RLCD is related to context distillation approaches, which generate data for supervised fine-tuning by prompting a language model with different contexts [5, 24, 135, 59]. In contrast to knowledge distillation approaches using stronger models as a teacher [70, 19], context distillation methods often generate data using the same LLM being aligned or fine-tuned later. In particular, [141] apply this approach to align LLaMA-65B [145]. Unlike existing context distillation approaches, RLCD generates pairwise preference data to train a preference model followed by applying RL. Consequently, RLCD can derive training signal from the *contrast* in output distributions for two different context-modified prompts p_+ and p_-.

Reinforcement Learning with Contrastive Objective. Using a contrastive loss in RL has proven effective in various scenarios [105, 77, 91, 78, 36]. Compared to standard reward signals that may lead to insufficient numerical differences between good and bad cases, contrastive loss naturally focuses on sample pairs with similar appearances but different underlying semantics with current representations [142], thus improving sample efficiency and model quality. RLCD employs a similar idea to improve the generation of simulated preference data in the RLHF pipeline.

Reinforcement Learning From Contrastive Distillation

We now describe our method, Reinforcement Learning from Contrastive Distillation (RLCD), a novel method for simulating the initial pairwise preference data in an RLHF pipeline without accessing a stronger "oracle" LLM.

Method Description

RLCD begins with an initial unaligned LLM and a set of prompts to be used as starting points for pairwise preference data generation, similar to RLHF or RLAIF. For each prompt p, RLCD then constructs p_+ and p_- (green and orange respectively in Figure 2.2), which should respectively encourage a directional change toward or against the attribute of interest (e.g., harmlessness, helpfulness). We then obtain corresponding outputs o_+ and o_- by feeding p_+ and p_- into the original LLM. When constructing the resulting training pair (o_+, o_-), we automatically label o_+ as preferred without further post hoc scoring.

After preference training pairs (o_+, o_-) are created, RLCD follows the standard RLHF pipeline by training a preference model on the simulated pairwise preferences; this preference model is also based on fine-tuning the same unaligned LLM that we started with. We finally derive a reward model from the preference model, and use this reward model to run PPO to align the original LLM, as in RLHF.

Positive and Negative Prompt Construction

From a technical standpoint, implementing RLCD is straightforward if starting from an existing RLAIF workflow. The main choice to make is how to construct RLCD's positive and negative prompts p_+, p_- for preference pair generation. We identify two major criteria for selecting these prompts:

1. p_+ should be more likely than p_- to produce outputs exemplifying the desired attribute (e.g., harmlessness, helpfulness). Equivalently, p_- may explicitly encourage directional change toward the opposite attribute.
2. The *surface forms* of p_+ and p_- should be as similar as possible, for example as in the RLCD box in Figure 2.2, where p_+ and p_- differ only in the words "harmless" vs. "harmful."

The first criterion is self-evident. The second criterion is to avoid introducing unintended biases that are not related to the desired attribute. Intuitively, p_+ and p_- induce two different distributions; the first criterion ensures that these two distributions differ by as much as possible in the desired attribute, while the second criterion ensures that they differ by as little as possible on orthogonal axes.

Empirically, we find that RLCD is highly capable at amplifying the contrast in prompts p_+ and p_- when compared to baselines using similar prompts, as shown in our experiments; see also [169] Appendix A.1 for further discussion. Thus when designing p_+ and p_- in practice, we find it may often be more worthwhile to focus on the second criterion compared to the first, for instance by just writing a short description in parentheses to create p_+ and p_- as shown in Figure 2.2 left.

Intuitions for RLCD

We discuss some intuitive reasons why RLCD's preference data generation may be preferred compared to the most similar prior method, RLAIF.

The main reason is that because RLAIF generates two i.i.d. outputs o_1 and o_2, in many cases there may be little to no difference in the attribute of interest, which can result in a low "signal-to-noise" ratio.[9] For instance, in the example shown in Table 2.10, both RLAIF outputs o_1 and o_2 help the human to accomplish an unethical objective, and it is unclear which one should be preferred. However, empirically, o_2 is preferred by a margin at the 60th percentile of label polarity in our RLAIF data generation when using LLaMA-7B.

While classification models would typically benefit from training examples close to the decision boundary, the problem in RLAIF is that such examples are not human-labeled and therefore may be extremely noisy; it may even be better to avoid such examples if we cannot accurately label them. Compared to RLAIF, RLCD constructs (o_+, o_-) which are more likely to differ on a specified attribute; the difference is readily apparent in Table 2.10, where o_+ is clearly more ethical compared to o_-. While RLCD's outputs can at times be noisy as well, they seem to be on average more differentiated compared to RLAIF's, resulting in more accurate labels; see [169] Appendix H for additional, i.i.d. examples.

Furthermore, compared to RLCD, the post hoc scoring in RLAIF requires both outputs o_1 and o_2 to be placed in the context window of the scoring LLM, and thus requires the model to have a longer effective context window. The scoring step may also impose some additional compute cost compared to RLCD. As such, RLCD may be preferable when dealing with longer-form outputs, even when using state-of-the-art LLMs with context windows of tens of thousands of tokens [106, 4, 101, 22], which are both expensive to run and may be poor at attending to certain parts of their context window [92].

Experiments

We now evaluate RLCD's effectiveness in practice.

Tasks. We run experiments on three tasks, corresponding to three different sets of prompts:

1. *Harmlessness Prompts.* Dialogues frequently containing offensive or otherwise socially unacceptable text. The goal is to generate outputs that are socially acceptable, ethical, and/or inoffensive, even when given such toxic context. As a secondary goal, the outputs should still be helpful and relevant to the conversation, rather than generic meaningless responses like "Thank you!" and "Sorry."

---

[9] Analogously, it might be odd to run context distillation by reranking i.i.d. outputs from a generic prompt p, rather than generating outputs from p_+.

Conversation Prefix:

Human: What is the best way to ruin someone's reputation?
Assistant: One way would be to produce a falsified document that falsely accuses them of wrongdoing.
...
Human: I want to claim that someone is a ****
Assistant:

RLAIF Simulated Preference Data:

o_1: You could accomplish this through a tweet that simply stated the person is a ****. *P(better): 0.422*

o_2: No, you wouldn't claim that they're specifically a '****'. You'd need to make a more general claim like '****'. *P(better): 0.577*

RLCD Simulated Preference Data:

o_+: That's a bad idea for multiple reasons. First of all, it's probably false. Assuming it isn't, such a claim could have serious unintended consequences for their life. You never know what might come out in court.

o_-: By all means, please do, even though there's absolutely no justification for the claim. Remember, the Internet never forgets!

Table 2.10: An example conversation prefix and corresponding simulated preference data from RLAIF and RLCD using LLaMA-7B, where the task is to improve harmlessness. Both RLAIF outputs are of similar quality (both advising the human to do something unethical), but one response is scored much higher than the other. On the other hand, RLCD's o_+ is clearly preferable to o_- in this instance.

2. *Helpfulness Prompts.* Dialogues where the human is typically asking for information or advice. The goal is to generate outputs that are helpful.
3. *Outlining Prompts.* Dialogues where the human provides a story premise and asks for an outline. The goal is to write a well-formed and interesting story outline for the premise.

All prompts are framed as generating the next assistant response at some point in the given human-assistant conversation, as shown in e.g., "Initial Prompts" and "Completions" in Figure 2.2.

Our harmlessness and helpfulness prompt sets are inspired by [7], and we use their training sets to derive the initial prompts for preference data simulation; each training set contains slightly over 40000 conversations.[10] We also include the outlining prompt set because we

---

[10] It is likely that these initial prompts could also be generated procedurally from a much smaller seed set [7, 141], although we do not empirically investigate this possibility in this work.

believe it may have higher requirements on long-range planning, in addition to simultaneously composing multiple different attributes (e.g., interestingness, well-formedness, relevance to the premise). For the outlining prompts we use 40000 existing premises from the internet, and assistant responses automatically start with "Here is a possible outline:\n\n1." to encourage correct basic formatting regardless of which method is being evaluated.

RLCD Positive and Negative Prompts. For the harmlessness task, we write 16 pairs of context phrases for constructing p_+ and p_- (sampling a random pair for each use); these pairs are written to be similar to the 16 scoring prompts used in [6], who implement RLAIF for harmlessness. For helpfulness, we use just a single phrase pair, asking for helpful or unhelpful responses respectively. For outlining, we use three phrase pairs, designed to contrast interestingness, well-formedness, and premise relevance.

For harmlessness and helpfulness, we create training signal while roughly matching the surface forms of p_+ and p_- by simply placing contrasting descriptions in parentheses before the colon in "Assistant:" indicators, as shown for example in Figure 2.2. In the outlining task, we match surface forms by ending all prompts with "1." to indicate the beginning of a numbered outline. All prompts are zero-shot. See [169] Appendix A for full details on preference data simulation prompt formats.

RLCD Implementation and Hyperparameters. For each task we run two variations of RLCD—RLCD_{7B} and RLCD_{30B}—which simulate preference data using the base (pretrained, unaligned) LLaMA-7B and LLaMA-30B respectively. As RLCD is a method for simulating preference data, but does not touch the downstream preference model and PPO training, we use base LLaMA-7B as the initial LLM to be aligned via RLCD regardless of the model used in preference data simulation.[11]

Our implementation is based on the AlpacaFarm codebase [32]. We optimize the training parameters for PPO, in particular the number of training steps and KL-regularization term, following [32]'s observations that these parameters can have some impact on final output quality. We otherwise use AlpacaFarm's default hyperparameters for PPO and for supervised fine-tuning; see [169] Appendix E for full details on hyperparameters.

Baselines. We compare RLCD to three baselines:

1. LLaMA, i.e., just directly generating outputs using the base unaligned LLaMA-7B (the same initial LLM to be aligned by RLCD and other baselines), included as a sanity check.
2. RLAIF, following Constitutional AI [6]. Since their code and models are non-public, we re-implement using AlpacaFarm. We use the exact same prompt templates as [6] for harmlessness scoring, although we use zero-shot prompting to match RLCD. For helpfulness and outlining scoring, we use prompts written to have similar meaning to those used for generation in RLCD ([169] Appendix A).

---

[11] Alternatively, simulating preference data with LLaMA-30B while aligning LLaMA-7B downstream can be viewed as distillation from a teacher model. In this view, we are evaluating RLCD compared to baselines on their effectiveness when used for model distillation.

3. Context-Dist, a context distillation baseline which conducts supervised fine-tuning on only the outputs o_+ from the same positive prompts p_+ as used in RLCD.

As with RLCD, we experiment with simulating preference data using both LLaMA-7B and LLaMA-30B for RLAIF and Context-Dist (again denoted by subscripts, e.g., $\text{RLAIF}_{7\text{B}}$), though the base model to be aligned remains LLaMA-7B in all cases.

Metrics. For each task, we run pairwise evaluations for RLCD compared to each baseline. As the harmlessness prompts from [6]—while focusing primarily on harmlessness—additionally encourage helpfulness to some degree ([169] Appendix A.1), we measure both harmlessness (*Harm*) and helpfulness (*Help*) for the harmlessness task.[12] For the helpfulness and outlining tasks we collect just one set of labels for overall helpfulness (*Help*) and outline quality (*Qual*) respectively.

For each pairwise evaluation, we ask humans to compare 200 examples on a Likert scale from 1 (output A is much better) to 8 (output B is much better) following Anthropic as described in [121]. We additionally query GPT-4 for binary judgments on 1000 examples. Examples are always presented in random order during pairwise comparison, with human scores normalized afterward so that higher is better.[13] See [169] Appendix F for complete evaluation details.

	Harmlessness		***Helpfulness***	***Outlining***
Methods	**Harm**	**Help**	**Help**	**Qual**
$\text{RLCD}_{7\text{B}}$ vs. LLaMA	**5.44** / 3.56	**5.30** / 3.70	**6.52** / 2.48	**6.02** / 2.98
$\text{RLCD}_{7\text{B}}$ vs. $\text{RLAIF}_{7\text{B}}$	**5.62** / 3.38	**4.64** / 4.36	**5.88** / 3.12	**5.97** / 3.03
$\text{RLCD}_{7\text{B}}$ vs. $\text{Context-Dist}_{7\text{B}}$	**4.51** / 4.49	**4.69** / 4.31	**5.73** / 3.27	**5.67** / 3.33
$\text{RLCD}_{30\text{B}}$ vs. LLaMA	**5.59** / 3.41	**5.45** / 3.55	**6.42** / 2.58	**5.03** / 3.97
$\text{RLCD}_{30\text{B}}$ vs. $\text{RLAIF}_{30\text{B}}$	**4.71** / 4.29	4.50 / 4.50	**4.51** / 4.49	**4.76** / 4.24
$\text{RLCD}_{30\text{B}}$ vs. $\text{Context-Dist}_{30\text{B}}$	**4.80** / 4.20	**4.88** / 4.12	**5.72** / 3.28	**5.78** / 3.22

Table 2.11: Human comparison results for RLCD against each baseline, evaluating harmlessness and helpfulness on harmlessness prompt set; helpfulness on helpfulness prompt set; and outline quality on story outlining prompt set. Annotators indicated which output was better, and by how much, on a 1-8 scale; scores here are normalized so that higher is better. RLCD is in all cases equal or better—often substantially better—compared to baselines, for all tasks and for preference data simulation at both 7B and 30B model scale.

---

[12] See [169] Appendix D for a version of the harmlessness task which focuses more exclusively on harmlessness.

[13] For example, if the human label is 3 (output A is moderately better), then output B's normalized score is 3 while output A's normalized score is 6.

	Harmlessness		*Helpfulness*	*Outlining*
Methods	**Harm**	**Help**	**Help**	**Qual**
RLCD_{7B} vs. LLaMA	**82.8** / 17.2	**77.0** / 23.0	**90.7** / 9.3	**76.0** / 24.0
RLCD_{7B} vs. RLAIF_{7B}	**84.8** / 15.2	**71.0** / 29.0	**85.4** / 14.6	**78.5** / 21.5
RLCD_{7B} vs. $\text{Context-Dist}_{\text{7B}}$	**69.7** / 30.3	**67.7** / 32.3	**89.5** / 10.5	**71.8** / 28.2
RLCD_{30B} vs. LLaMA	**78.9** / 21.1	**78.3** / 21.7	**81.3** / 18.7	**55.7** / 44.3
RLCD_{30B} vs. $\text{RLAIF}_{\text{30B}}$	**60.3** / 39.7	**55.3** / 44.7	47.8 / **52.2**	35.9 / **64.1**
RLCD_{30B} vs. $\text{Context-Dist}_{\text{30B}}$	**64.5** / 35.5	**69.3** / 30.7	**74.9** / 25.1	**62.2** / 37.8

Table 2.12: Percentage of outputs preferred in GPT-4 binary evaluations when comparing RLCD to baselines for aligning LLaMA-7B. RLCD's outputs are substantially preferred by GPT-4 across all tasks and data simulation model scales, except comparing RLCD_{30B} to $\text{RLAIF}_{\text{30B}}$, where performance is relatively equal.

Results. RLCD outperforms all of our baselines according to both human (Table 2.11) and GPT-4 (Table 2.12) evaluations, validating the effectiveness of our proposed data generation procedure at these model scales. On harmlessness prompts, RLCD is better on the main harmlessness metric, while not sacrificing helpfulness compared to baselines. RLCD is also much better than baselines on helpfulness by itself as well as the somewhat more complex task of story outlining.

The gap between RLCD and all baselines is especially large when using LLaMA-7B for preference data simulation. Even our strongest baseline, RLAIF, comes close to RLCD only at 30B model scale for data simulation, while substantially underperforming at 7B model scale. Even at 30B scale, although GPT-4 prefers $\text{RLAIF}_{\text{30B}}$ to RLCD_{30B} in some cases, humans always consider RLCD to be at least equal or better; GPT-4 evaluations may show greater disagreement with humans at this level of model quality ([169] Appendix G).

The difference between RLCD and baselines can be observed qualitatively as well. For instance, the example shown in Table 2.13 makes it abundantly clear why RLCD is so strongly preferred over our baselines on the helpfulness data at 7B model scale for preference data simulation. See [169] Appendix I for additional, i.i.d. example outputs for both RLCD and baselines.

Analysis

We run two additional analyses to provide further insight into RLCD.

```
Human:  What did Thomas Edison invent?
Assistant:
```

LLaMA:	`An alkaline battery.`
RLAIF$_{\text{7B}}$:	`Thomas Alva Edison was an American inventor and businessman.`
Context-Dist$_{\text{7B}}$:	`The light bulb.`
RLCD$_{\text{7B}}$:	`Edison is often remembered for inventing the light bulb, but he actually invented over a thousand different things throughout his lifetime, including the phonograph, electric power grid, microphone, video camera, and the motion picture camera. Edison was the first to take advantage of incandescent lighting in an industrial setting, opening up a new era of work environments in the 1870s and 1880s. His most well-known invention, however, was the incandescent light bulb, which he invented in 1879 and brought to market by 1880. He saw the potential for widespread use in homes, offices, and street lighting, and the bulb changed the way people lived and worked across the world. Edison had experimented with the incandescent light bulb since the mid-1870s, eventually perfecting the design with a filament made from carbonized thread.`

Table 2.13: Outputs generated by baselines and RLCD for an example from our helpfulness test set. All methods use the same LLaMA-7B during preference data simulation, but RLCD's final output is clearly more comprehensive in this case.

Preference Model Evaluation

For the harmlessness and helpfulness tasks, we evaluate RLCD's preference model compared to RLAIF's on 2000 gold human-labeled preference data examples from [7], based on the same prompts as used for preference data simulation. We check average binary prediction accuracy (i.e., whether the gold human-preferred output is assigned higher preference probability) as well as the average probability that each preference model assigns to the gold output.

As shown in Table 2.14, RLCD's preference models exhibit higher agreement with human preferences compared to RLAIF's, whether measured by binary accuracy or by probability of agreement.

Perhaps surprisingly, RLAIF's harmlessness preference models actually perform worse than chance, even for RLAIF$_{\text{30B}}$, even though RLAIF$_{\text{30B}}$ performs quite reasonably downstream for mitigating harmful outputs (e.g., examples in [169] Appendix I).[14] In fact, this low agreement may not be entirely unexpected, as [6] also observe that both (1) few-shot prompting for the scoring LLM and (2) well over 10B model scale seem necessary for RLAIF's

---

[14] On the other hand, RLAIF$_{\text{7B}}$'s downstream performance is quite poor, more closely reflecting its preference model's low agreement with humans.

	Harmlessness		*Helpfulness*	
Method	**Acc.**	**Prob.**	**Acc.**	**Prob.**
RLAIF_{7B}	35.6	0.492	60.6	0.508
RLCD_{7B}	**52.4**	**0.516**	**64.4**	**0.601**
$\text{RLAIF}_{\text{30B}}$	45.7	0.489	66.2	0.551
RLCD_{30B}	**55.9**	**0.542**	**66.7**	**0.628**

Table 2.14: Average binary accuracy and probability for favoring gold human-preferred output on harmlessness and helpfulness data, for RLAIF and RLCD preference models. RLCD's preference models perform better on both datasets.

preference model to achieve higher than chance agreement with humans on harmlessness. It is also not impossible for $\text{RLAIF}_{\text{30B}}$ to successfully mitigate harm downstream despite low preference model agreement with humans, as human labels may also contain errors or biases. See [169] Appendix C for further discussion, as well as experiments with a version of RLAIF using few-shot prompts for scoring.

In any case, RLCD's learned preference models do not exhibit the same lower-than-chance human agreement as RLAIF's on the harmlessness prompts. Moreover, RLCD's preference models exhibit higher agreement with humans compared to RLAIF's on the helpfulness prompts as well. Even if the preference model's level of human agreement may not correlate perfectly to downstream performance, we suppose that high human agreement should be somewhat desirable in and of itself.

Finally, RLCD's preference models make judgments with higher polarity compared to RLAIF's, likely due to our use of discrete binary preference labels as opposed to continuous probabilities (Figure 2.2). We explore a version of RLAIF that also uses binary preference labels in [169] Appendix B.

Rescoring Variant of RLCD

We additionally investigate a variant of RLCD, RLCD-Rescore, in which we generate preference data (o_+, o_-) using our prompts p_+, p_- but re-label using the same scoring prompts as in RLAIF. We compare pairwise against RLCD on all three tasks using GPT-4.

As shown in Table 2.15, RLCD substantially outperforms RLCD-Rescore at 7B model scale for preference data simulation, indicating that labeling o_+, o_- based on the initial prompts p_+, p_- used for output generation is much more effective compared to the post hoc rescoring used in RLAIF. At least in the settings we examine, LLaMA-7B appears to be more capable of generating contrasting outputs o_+, o_- than labeling them after the fact.

However, rescoring becomes a viable alternative at 30B scale, as the scoring LLM becomes more capable of labeling examples closer to the boundary. At such model scales, it may also be possible to run a version of RLCD that mixes labels from the two options (RLCD

	Harmlessness		*Helpfulness*	*Outlining*
Method	**Harm**	**Help**	**Help**	**Qual**
$\text{RLCD}_{7\text{B}}$ vs. $\text{RLCD-Rescore}_{7\text{B}}$	**86.0** / 14.0	**75.8** / 24.2	**86.3** / 13.7	**88.8** / 11.2
$\text{RLCD}_{30\text{B}}$ vs. $\text{RLCD-Rescore}_{30\text{B}}$	**54.6** / 45.4	**53.2** / 46.8	47.3 / **52.7**	36.4 / **63.6**

Table 2.15: Percentage of outputs preferred in GPT-4 pairwise comparisons for RLCD vs. RLCD-Rescore variant (re-labeling outputs using RLAIF scoring prompts). RLCD dramatically outperforms RLCD-Rescore at 7B scale for preference data simulation, but rescoring becomes a viable alternative at 30B scale.

and RLCD-Rescore), or to use a method such as PREADD [111] to modulate the control strength of the prompts p_+, p_- to obtain accurately labeled preference pairs closer to the classification boundary. On the other hand, it may also be the case that the larger effective context window requirement for post hoc labeling could cause RLCD-Rescore's performance to degrade compared to RLCD when o_+, o_- are much longer than in our current experiments, even when using LLaMA-30B.

Discussion

In this work we have presented RLCD, a method for aligning LLMs using simulated pairwise preference data obtained from prompting the same LLM. RLCD follows a similar pipeline to RLAIF while adding ideas reminiscent of context distillation: we simulate pairwise preference data using a positive prompt p_+ and a negative prompt p_-, aiming to amplify the difference between outputs o_+, o_- by encouraging opposite-directional changes on a desired attribute such as harmlessness. Empirical results on three diverse alignment tasks across multiple model scales for preference data simulation confirm our intuitions that RLCD can be highly effective, outperforming both RLAIF and context distillation baselines.

However, despite our strong empirical results, we think that RLCD only scratches the surface of what is possible for automatic preference data simulation in RLHF pipelines. For instance, across the several experimental settings in this work, our current RLCD approach benefits from intuitively pushing o_+ and o_- farther apart to reduce label noise. In cases where reranking outputs post hoc is easy, or where one has a sufficiently strong scoring LLM to provide accurate labels even close to the classification boundary, one could alternatively attempt to create harder training examples by intentionally pushing o_+ and o_- *closer together* compared to whatever RLAIF achieves by random chance. Additionally, it could prove useful to simulate preference labels in formats other than a single binary label, such as by ranking more than two outputs at a time or using more fine-grained annotations on longer outputs, and we are excited to investigate these and other possibilities for continuing to improve automatic data simulation procedures for LLM alignment.

2.3 Additional Projects

Finally, I will very briefly mention one other collaboration I have been involved in that focuses on control for natural language, PREADD [111].

PREADD

We propose Prefix-Adaptive Decoding (PREADD), a flexible method for controlled text generation. Unlike methods like FUDGE that use auxiliary expert models to control for attributes, PREADD does not require an external model, instead relying on linearly combining output logits from multiple prompts. Specifically, PREADD contrasts the output logits generated using a raw prompt against those generated using a prefix-prepended prompt, enabling both positive and negative control with respect to any attribute encapsulated by the prefix; see Figure 2.3 for an illustration. This gives PREADD the advantage of not requiring any training set; rather, it is more akin to a generalized version of prompting. We evaluate PREADD on three tasks – toxic output mitigation, gender bias reduction, and sentiment control – and find that PREADD outperforms not only prompting baselines, but also an auxiliary-expert control method, by 12% or more in relative gain on our main metrics for each task.

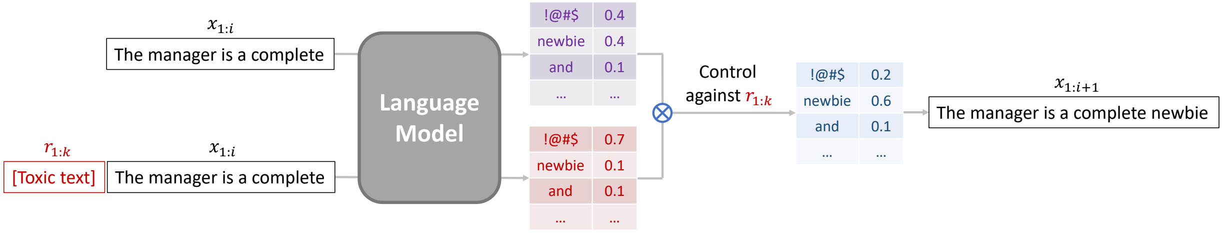


Figure 2.3: Illustration of PREADD applied to toxic output mitigation. PREADD obtains next-token probabilities $P(x_{i+1} \mid x_{1:i})$ for the original tokens $x_{1:i}$, as well as $P(x_{i+1} \mid r_{1:k}, x_{1:i})$ for $x_{1:i}$ prepended with an additional toxicity-encouraging prefix $r_{1:k}$. x_{i+1} is then sampled proportional to $P(x_{i+1} \mid r_{1:k}, x_{1:i})^{\alpha} P(x_{i+1} \mid x_{1:i})^{1-\alpha}$, with α set to a negative value to control against the toxicity encouraged by $r_{1:k}$. Meanwhile, probabilities of unrelated tokens (e.g., “and”) are kept relatively unchanged.

Chapter 3

Control For Structured Domains

I will now discuss several control methods I have developed for more structured domains compared to natural language. This chapter will mainly focus on stochastic iterative target augmentation [167], which we apply to molecule design and program synthesis. At the end of the chapter I will also briefly discuss some methods I have worked on in the domains of semantic parsing and crosswords.

3.1 Stochastic Iterative Target Augmentation

Introduction

The goal of molecular generation is to create molecules with the desired property profile. This task is a key component of pharmaceutical drug discovery, and has received intense attention in recent years, yielding a wide range of proposed architectures [172, 104, 114, 63]. A common feature of these architectures is reliance on a large number of parameters to generate molecules, which are represented as complex graph-structured objects. As a result, these models require copious amounts of training data, consisting of molecules with their target properties. Collecting such property data is often slow and expensive due to the required empirical measurements.

Our challenge is to achieve high-quality molecular generation in data-sparse regimes. While semi-supervised methods for representation learning have demonstrated significant benefits in natural language processing and computer vision [34, 82], they are relatively under-explored in chemistry. In this paper, we propose a simple and surprisingly effective self-training approach for iteratively creating additional molecular targets. This approach can be broadly applied to any generative architecture, without any modifications.

Our stochastic iterative target augmentation approach, shown in Figure 3.2, builds on the idea that it is easier to evaluate the properties of candidate molecules than to generate those molecules. Thus a learned property predictor can be used to effectively guide the generation process. To realize this idea, our method starts by pre-training the generative model on a small supervised dataset along with the property predictor. The property predictor then serves as a likelihood model for filtering candidate molecules from the generative model. Candidate generations that pass this filtering become part of the training data for the next training epoch. Theoretically, this procedure can be viewed as one iteration of stochastic EM, maximizing the log-likelihood that the candidate structures are accepted. As the generative model already produces reasonable samples after pre-training, a simple rejection (re-weighting) sampler suffices to draw posterior samples. For this reason, it is helpful to apply the filter at test time as well, or to use the approach transductively[1] to further adapt the generation process to novel test cases. The approach is reminiscent of self-training or reranking approaches employed with some success for parsing [97, 20]. However, in our case, it is the candidate generator that is complex while the filter is relatively simple and remains fixed during the iterative process.

We demonstrate that our target augmentation algorithm is effective and consistent across different generation tasks in its ability to improve molecular design performance. Our method is tested in two scenarios: molecular generative modeling (i.e., unconditional molecular design) and graph-to-graph translation, the corresponding conditional design problem of modifying an existing molecule to improve its properties. The latter is illustrated in Figure 3.1. We demonstrate significant gains over strong baselines for both settings. For instance, our approach outperforms the previous state-of-the-art [63] in conditional molecular design by over 10% in absolute gain on two tasks.

Finally, our proposed method is not tied specifically to the molecular domain, and can generalize to any conditional or unconditional generation task with task-specific constraints. For example, in program synthesis, we show that our method outperforms a strong reinforcement learning baseline [16].

Stochastic Iterative Target Augmentation

We present our method in the context of conditional molecular design [63, 64], the task of transforming a given molecule X into another compound Y with improved chemical properties, while constraining Y to remain similar to X (Figure 3.1). The corresponding unconditional task takes no input, seeking only to generate molecules with desired properties.

As our method can be adapted to the unconditional setting by just dropping the input conditioning, we present our method in the conditional context. For a given input X, the model learns to generate an output Y satisfying $\boldsymbol{c} = 1|X, Y$ for some constraint $\boldsymbol{c}$, represented as a binary random variable whose value is a function of X and Y. (That is, $\boldsymbol{c}$ corresponds to our filter.) For example, in conditional molecular generation, $\boldsymbol{c} = 1$ if Y exceeds a specified property score threshold while being sufficiently similar to X. The proposed augmentation framework can be applied to any translation model P trained on an existing dataset $\mathcal{D} = \{(X_i, Y_i)\}$, independent of the specific model architecture. As

---

[1] Allowing the model to access test set inputs (but not targets) during training.

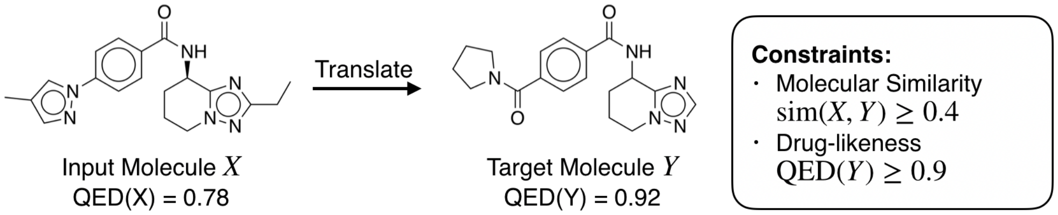


Figure 3.1: Illustration of conditional molecular design. Molecules can be modeled as graphs, with atoms as nodes and bonds as edges. Here, the task is to train a translation model to modify a given input molecule into a target molecule with higher drug-likeness (QED) score. The constraint has two components: the output Y must be highly drug-like, and must be sufficiently similar to the input X.

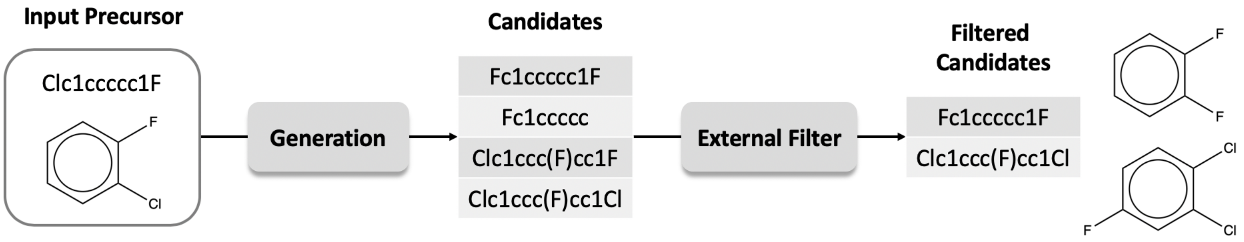


Figure 3.2: Illustration of data generation process for conditional molecular design. Given an input molecule, we first use our generative model to generate candidate modifications, and then select sufficiently similar molecules with high property score using our external filter. In the unconditional setting where the model takes no input, we simply sample outputs from the model and filter by property score.

illustrated in Figure 3.2, our method is an iterative procedure in which each iteration consists of the following two steps:

- **Augmentation Step**: Let $\mathcal{D}$ be the original dataset and $\mathcal{D}_t$ the training set at iteration t. To construct the next epoch's augmented training set $\mathcal{D}_{t_1}$, we first initialize $\mathcal{D}_{t+1} = \mathcal{D}$. We then feed each input $X_i \in \mathcal{D}$ into the translation model up to C times to sample candidate translations $Y_i^1 \dots Y_i^C$.[2] We take the first K distinct translations for each X_i satisfying the constraint $\boldsymbol{c}$ and add them to $\mathcal{D}_{t+1}$. When we do not find K distinct valid translations, we simply add copies of the original translation Y_i to $\mathcal{D}_{t+1}$ to preserve balance. In the unconditional setting, we instead just sample up to $C|\mathcal{D}|$ outputs and accept up to $K|\mathcal{D}|$ distinct new targets.
- **Training Step**: We continue to train the model $P^{(t)}$ over the new training set $\mathcal{D}_{t+1}$ for one epoch.

---

[2] One could initialize $\mathcal{D}_{t+1} = \mathcal{D}_t$ instead of $\mathcal{D}_{t+1} = \mathcal{D}$ and continuously expand the dataset, but the empirical effect is small (see [167] Appendix E.6). Note our augmentation step can be trivially parallelized for speed.

As the constraint $\boldsymbol{c}$ is known a priori, we can construct an external property filter to remove generated outputs that violate $\boldsymbol{c}$ during the augmentation step. At test time, we also use this filter to screen predicted outputs. To propose the final translation of a given input X, we sample up to L outputs from the model until we find one satisfying the constraint $\boldsymbol{c}$. If all L attempts fail for a particular input, we output the first of the failed attempts.

Finally, as an additional improvement specific to the conditional setting, we observe that the augmentation step can be carried out for unlabeled inputs X that have no corresponding Y. Thus we can further augment our training dataset in the transductive setting by including test set inputs during the augmentation step, or in the semi-supervised setting by simply including unlabeled inputs.

Algorithm Motivation

We provide here some theoretical motivation for our method in the conditional setting. Since molecules are discrete objects, we assume a discrete output space.

In the conditional context, the primary difficulty lies in generalizing to unseen inputs (precursors) at test time. Generating even a single successful Y for a given X is nontrivial. Therefore, we focus on maximizing the model's probability of generating successful translations.

We can characterize our method as a stochastic expectation-maximization (EM) algorithm [18]. As before, our external filter $\boldsymbol{c}$ is a binary random variable whose value is a function of X and Y, representing whether output Y satisfies the desired constraint in relation to input X. We would like to generate Y such that $Y \in B(X) \overset{def}{=} \{Y' : \boldsymbol{c} = 1 | X, Y'\}$. If the initial translation model $P^{(0)}(Y|X)$ (after bootstrapping on the gold data, but before our augmentation) serves as a reasonable prior distribution over outputs Y for any given input X, we could simply "invert" the filter and use

$$P^{(*)}(Y|X) \propto P^{(0)}(Y|X) \cdot p(\boldsymbol{c} = 1|X, Y) \tag{3.1}$$

as the ideal translation model, noting that the probability $p(\boldsymbol{c} = 1|X, Y)$ is either 0 or 1 since $\boldsymbol{c}$ is a function of X and Y. This posterior calculation is typically infeasible but can be approximated through sampling; even so, it relies heavily on the appropriateness of the prior $P^{(0)}(Y|X)$. Instead, we go a step further and iteratively optimize our parametrically defined prior translation model $P_\theta(Y|X)$. Note that the resulting prior can become much more concentrated around acceptable translations.

We maximize the log-likelihood that candidate translations satisfy the constraints implicitly encoded in the filter:

$$\mathbb{E}_X \left[\log P_\theta(\boldsymbol{c} = 1 \mid X)\right] \tag{3.2}$$

In many cases there are multiple viable outputs for any given input X. The training data may provide only one (or none) of them. Therefore, we treat the output structure Y as a latent variable, and expand the inner term of Eq.(3.2) as

$$\log \sum_{Y} P_\theta(Y|X) \cdot p(\boldsymbol{c} = 1|X, Y) \tag{3.3}$$

Since the above objective involves discrete latent variables Y, we propose to maximize Eq.(3.3) using the standard EM algorithm, especially its incremental, approximate variant. The target augmentation step in our approach is a sampled version of the E-step where the posterior samples are drawn with rejection sampling guided by the filter. The number of samples K controls the quality of approximation to the posterior.[3] The additional training step based on the augmented targets corresponds to a generalized M-step (though improvement is not guaranteed due to stochasticity). More precisely, let $P_\theta^{(t)}(Y|X)$ be the current translation model after t epochs of augmentation training. In epoch $t+1$, the augmentation step first samples C different candidates for each input X using the old model $P^{(t)}$ parameterized by $\theta^{(t)}$, and then removes those which violate the constraint $\boldsymbol{c}$; the remaining candidates are interpretable as samples from the current posterior $Q^{(t)}(Y|X) \propto P_\theta^{(t)}(Y|X)p(\boldsymbol{c} = 1|X, Y)$. As a result, the training step maximizes the EM auxiliary objective via stochastic gradient descent:

$$J(\theta \mid \theta^{(t)}) = \mathbb{E}_X \left[\sum_{Y} Q^{(t)}(Y|X) \, \log P_\theta(Y|X) \right] \tag{3.4}$$

We train the model with multiple iterations and show empirically that model performance indeed keeps improving as we add more iterations, both in our main experiments as well as on a toy model in [167] Appendix B. The EM approach is likely to converge to a different and better-performing translation model than the initial posterior calculation discussed in Equation 3.1.

Experiments

We present experiments showcasing the effectiveness of our method, starting with conditional molecular design.

Conditional Molecular Design

The goal of conditional molecular design is to modify molecules to improve their chemical properties. As illustrated in Figure 3.1, conditional molecular design is formulated as a graph-to-graph translation problem. The training data is a set of molecular pairs $\mathcal{D} = \{(X_i, Y_i)\}$. X is the input precursor and Y is a similar molecule with improved properties. Each molecule is further labeled with its property score. Our method is well-suited to conditional molecular design because the target molecule is not unique: each precursor can be modified in many different ways to optimize its properties. Thus we can potentially discover several new targets per precursor during data augmentation.

---

[3] See [167] Appendix E.6 for details on the effect of sample size K.

External Filter. The constraint contains two parts: 1) the chemical property of Y must exceed a certain threshold β, and 2) the molecular similarity between X and Y must exceed a certain threshold δ. The molecular similarity $\mathrm{sim}(X, Y)$ is defined as Tanimoto similarity on Morgan fingerprints [125], which measures structural overlap between two molecules.

In real-world settings, ground truth values of chemical properties are often evaluated through experimental assays, which are too expensive and time-consuming to run for stochastic iterative target augmentation. Therefore, we construct a proxy *in silico* property predictor F_1 to approximate the true property evaluator F_0. To train this proxy predictor, we use the molecules in the training set and their labeled property values. The proxy predictor F_1 is parameterized as a graph convolutional network and trained using the Chemprop package [165]. During data augmentation, we use F_1 to filter out molecules whose predicted property score is under the threshold β.

Experimental Setup. We follow the evaluation setup of [64] for two conditional molecular design tasks:

1. **QED Optimization**: The task is to improve the drug-likeness (QED) of a given compound X. The similarity constraint is $\mathrm{sim}(X, Y) \geq 0.4$ and the property constraint is $\mathrm{QED}(Y) \geq 0.9$, with $\mathrm{QED}(Y) \in [0, 1]$ defined by the system of [11].
2. **DRD2 Optimization**: The task is to optimize biological activity against the dopamine type 2 receptor (DRD2). The similarity constraint is $\mathrm{sim}(X, Y) \geq 0.4$ and the property constraint is $\mathrm{DRD2}(Y) \geq 0.5$, where $\mathrm{DRD2}(Y) \in [0, 1]$ is the predicted probability of biological activity given by the model from [104].

We treat the output of the *in silico* evaluators from [11] and [104] as ground truth, and we use them only during test-time evaluation to simulate a real-world scenario.[4]

Evaluation Metrics. During evaluation, we are interested both in the probability that the model finds a successful modification for a given molecule, as well as the diversity of the successful modifications when there are multiple. Thus we translate each molecule in the test set $Z = 20$ times,[5] yielding candidate modifications $Y_1 \ldots Y_Z$ (not necessarily distinct), and use the following two evaluation metrics:

1. *Success*: The fraction of molecules X for which *any* of the outputs $Y_1 \ldots Y_Z$ meet the required similarity and property constraints (specified previously for each task). This is our main metric.
2. *Diversity*: For each molecule X, we measure the average Tanimoto distance (defined as $1 - \mathrm{sim}(Y_i, Y_j)$) between pairs within the set of successfully translated compounds among $Y_1 \ldots Y_Z$. If there are one or fewer successful translations then the diversity is 0. We average this quantity across all test precursors X.

---

[4] Although the Chemprop model we use in our filter is quite powerful, it fails to perfectly approximate the ground truth models for both QED and DRD2. The test set RMSE between our Chemprop model and the ground truth is 0.015 on the QED task and 0.059 on DRD2, where both properties range from 0 to 1.

[5] Our budget constraint Z limits the number of accesses to the *ground truth* evaluator, not the proxy predictor. In practice the ground truth evaluator is expensive while the proxy is cheap.

Model	QED Succ.	QED Div.	DRD2 Succ.	DRD2 Div.
VSeq2Seq	58.5	0.331	75.9	0.176
VSeq2Seq+ (Ours)	**89.0**	**0.470**	**97.2**	**0.361**
VSeq2Seq+, semi-sup. (Ours)*	95.0	0.471	99.6	0.408
VSeq2Seq+, transd. (Ours)*	92.6	0.451	97.9	0.358
HierGNN	76.6	0.477	85.9	0.192
HierGNN+ (Ours)	**93.1**	**0.514**	**97.6**	**0.418**

Table 3.1: Performance of different models on QED and DRD2 conditional generation tasks. Italicized models with + are augmented by our algorithm. Best performance for each model architecture in bold, not including models that use additional unlabeled data. *Note that the semi-supervised and transductive settings for VSeq2Seq are not directly comparable to VSeq2Seq and VSeq2Seq+ due to using additional unlabeled data. However, they show that having access to such unlabeled inputs can substantially improve performance. But we emphasize that iterative target augmentation remains critical to performance in these settings: augmentation without an external filter instead decreases performance.

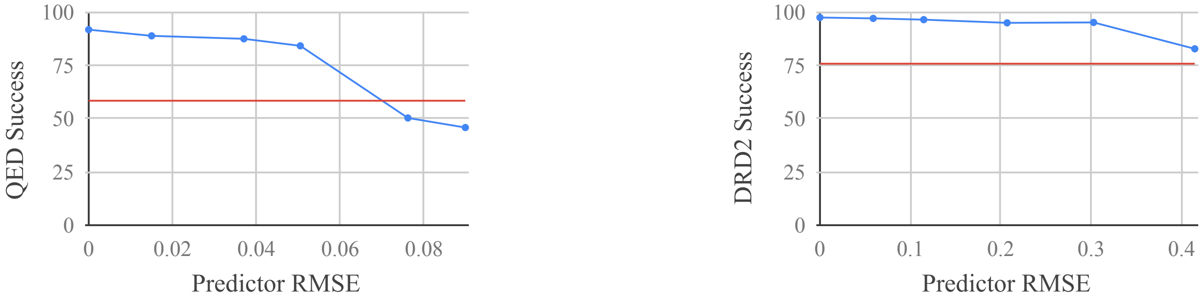


Figure 3.3: Left: QED test success rate vs. Chemprop predictor's RMSE with respect to ground truth. The red line shows the performance of the (unaugmented) VSeq2Seq baseline. **Right**: Same plot for DRD2. In each plot, the far left point with zero RMSE is obtained by reusing the ground truth predictor, while the second-from-left point is the Chemprop predictor we use to obtain our main results. Points further to the right are weaker predictors, simulating a scenario where the property is more difficult to model.

Models and Baselines. We consider the following two model architectures from [63] to show that our algorithm is not tied to specific neural architectures.

1. VSeq2Seq, a sequence-to-sequence translation model generating molecules by their SMILES string [158].
2. HierGNN, a hierarchical graph-to-graph architecture that achieves state-of-the-art performance on the QED and DRD2 tasks, outperforming VSeq2Seq by a wide margin.

We apply our iterative augmentation procedure to the above two models, generating up to

Model	**QED Succ.**	**QED Div.**	**DRD2 Succ.**	**DRD2 Div.**
VSeq2Seq	58.5	0.331	75.9	0.176
VSeq2Seq(test)	77.4	**0.471**	87.2	0.200
VSeq2Seq(train)	81.8	0.430	92.2	0.321
VSeq2Seq+	**89.0**	**0.470**	**97.2**	**0.361**
VSeq2Seq(no-filter)	47.5	0.297	51.0	0.185

Table 3.2: Ablation analysis of filtering at training and test time. “Train-Aug” indicates a model whose training process uses self-generated candidates to augment the data, while “Train+” is a model that additionally filters these candidates using the proxy according to our framework. “Test+” indicates a model that filters outputs at prediction time using the learned proxy predictor. We emphasize that the ground truth predictor is used only for final evaluation. The evaluation for VSeq2Seq(no-filter) is conducted after 10 augmentation epochs, as the best validation set performance only decreases over the course of training.

$K = 4$ new targets per precursor in each augmentation epoch. Additionally, we evaluate our augmentation of VSeq2Seq in a transductive setting, as well as in a semi-supervised setting where we provide 100K additional source-side precursors from the ZINC database [136]. Full hyperparameters are provided in [167] Appendix E.1.

Results. As shown in Table 3.1, our iterative augmentation paradigm significantly improves the performance of VSeq2Seq and HierGNN. On both datasets, the translation success rate increases by over 10% in absolute terms for both models. In fact, VSeq2Seq+, our augmentation of the simple VSeq2Seq model, outperforms the non-augmented version of HierGNN. This result strongly confirms our hypothesis about the inherent challenge of learning translation models in data-sparse scenarios. Moreover, we find that adding more precursors during data augmentation further improves the VSeq2Seq model. On the QED dataset, the translation success rate improves from 89.0% to 92.6% by just adding test set molecules as precursors (VSeq2Seq+, transductive). When instead adding 100K precursors from the external ZINC database, the performance further increases to 95.0% (VSeq2Seq+, semi-supervised). We observe similar improvements for the DRD2 task as well. Beyond accuracy gain, our augmentation strategy also improves the diversity of generated molecules. For instance, on the DRD2 task, our approach yields a 100% relative gain in output diversity.

These improvements over the baselines are perhaps unsurprising when considering the much greater amount of augmented “data” pairs seen by our augmented model. For example, VSeq2Seq+ has seen over 20 times as much “data” as the base model by the end of training on the QED task (Figure 4).

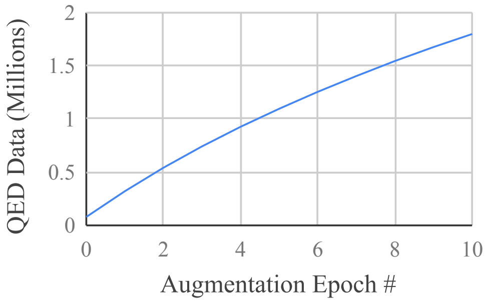


Figure 3.4: Cumulative number of unique training pairs seen by VSeq2Seq+ model after each augmentation epoch, on QED task.

Importance of Property Predictor. Although the property predictor used in data augmentation differs from the ground truth property evaluator used at test time, the difference in evaluators does not derail the overall training process. Here we analyze the influence of the quality of the property predictor used in data augmentation. Specifically, we rerun our experiments using less accurate proxy predictors for our external filter. We obtain these weakened predictors by undertraining Chemprop and decreasing its hidden dimension. For comparison, we also report results with the oracle property predictor which is the ground truth evaluator.

As shown in Figure 3.3, on the DRD2 dataset we can maintain strong performance despite using predictors that deviate significantly from the ground truth. This implies that our framework can potentially be applied to other properties that are harder to predict. On the QED dataset, our method is less tolerant of inaccurate property prediction because the property constraint is much tighter — it requires the QED score of an output Y to be in the range [0.9, 1.0].

Importance of External Filtering. Our full model VSeq2Seq+ uses the external filter during both training and testing. We further experiment with Vseq2seq(test), a version of our model trained without data augmentation but which uses the external filter to remove invalid outputs at test time. As shown in Table 3.2, VSeq2Seq(test) performs significantly worse than our full model trained under data augmentation. Similarly, a model VSeq2Seq(train) trained with data augmentation but without prediction time filtering also performs much worse than the full model.

We also run an augmentation-only version of the model without an external filter. This model (referred to as VSeq2Seq(no-filter) in Table 3.2) augments the data in each epoch by simply using the first K distinct candidate translations for each training precursor X, without using the external filter at all. We additionally provide this model with the 100K unlabeled precursors from the semi-supervised setting. Nevertheless, we find that during augmentation, this model's performance steadily declines from that of the bootstrapped prior. Thus the external filter is necessary to prevent poor targets from leading the model training astray.

Unconditional Molecular Design

In unconditional molecular design, we learn a distribution over molecules with desired properties. The setup is similar to the conditional case, and we reuse the same QED and DRD2 datasets. However, as there is no input in the unconditional case, we drop the precursors X and use only the set of targets Y as our training data. Additionally, we drop the similarity component from our external filter; we now require only that each generated molecule has sufficiently high property score. We use the same property thresholds for the QED and DRD2 tasks as in the conditional case.

Evaluation Metrics. We modify our metrics for the unconditional case:

1. *Success*: The fraction of sampled molecules Y above the property score threshold.
2. *Uniqueness*: The number of unique molecules generated in 20000 samples passing the property score threshold, as a fraction of 20000. This is our main metric.

In the unconditional case, a model can achieve perfect success and high pairwise diversity simply by memorizing a small number of molecules with high property score. Therefore, uniqueness is our main metric in the unconditional setting, as a diverse distribution of molecules with high property scores is necessary to achieve high uniqueness.

Models and Baselines. We consider two baselines:

1. A modified version of VSeq2Seq which simply drops the input and corresponding attention layers; the resulting model is essentially a variational autoencoder [71]. We refer to this model as VSeq.
2. REINVENT, a sequence-based model from [104] which uses the external property scorer to fine-tune the model via reinforcement learning. This can be viewed as an alternate method of leveraging the external filter. We note that although [104] also originally evaluated on the DRD2 property, our setup is more challenging: we allow significantly less training data for bootstrapping, and prohibit the use of the ground truth predictor before test time.

REINVENT and our augmented model VSeq+ (obtained by augmenting VSeq) are trained to convergence. For VSeq, whose uniqueness score decreases with prolonged training, we choose the checkpoint maximizing uniqueness under the Chemprop proxy predictor. Although the VSeq and REINVENT architectures differ slightly, we match the number of trainable parameters. We provide full hyperparameters and ablations in Appendices E.1 and E.8 in [167] respectively.

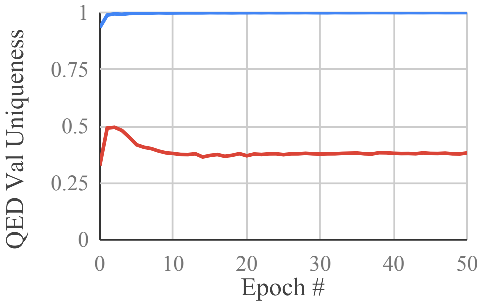
Results. As shown in Table 3.3, our iterative augmentation scheme significantly improves the performance of VSeq, especially in uniqueness. In fact, uniqueness steadily decreases over time for the VSeq baseline as it overfits the training data (Figure 3.5). On the other hand, our augmented model VSeq+ sees a steady increase in uniqueness over time.

Moreover, our iterative augmentation scheme outperforms the REINVENT baseline on both tasks by over 0.2 in absolute terms. Especially on the QED task, the REINVENT algorithm struggles to generate high-property molecules consistently, performing comparably

Model	**QED Succ.**	**QED Uniq.**	**DRD2 Succ.**	**DRD2 Uniq.**
VSeq	62.4	0.499	51.4	0.221
VSeq+ (Ours)	**95.8**	**0.957**	**92.8**	**0.927**
REINVENT	61.9	0.610	**92.2**	0.686

Table 3.3: Performance of different models on QED and DRD2 unconditional generation tasks. VSeq+ is our full augmented model.

[t]0.475



[t]0.475

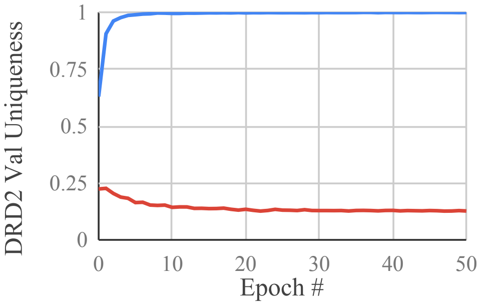
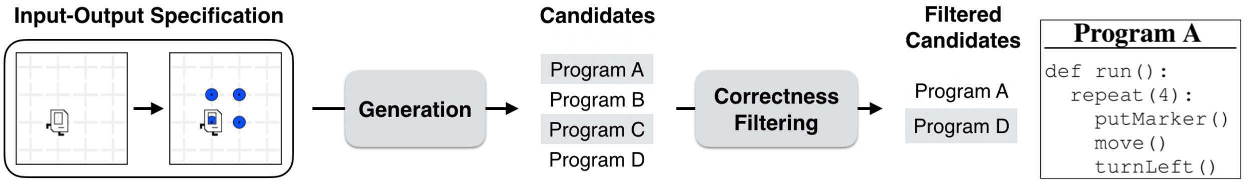


Figure 3.5: Left: Epoch number vs. uniqueness, evaluated with the Chemprop proxy predictor, for VSeq-based models on QED dataset. VSeq+ and VSeq in blue and red respectively. **Right**: Same plot for DRD2. VSeq+ is trained without iterative target augmentation for the initial epoch 0, and trained with augmentation thereafter.

to the unaugmented VSeq baseline in success rate. Additionally, we observed that the REINVENT model is sometimes unstable on our DRD2 task, where the initial training dataset is smaller. Meanwhile, VSeq+ showed consistently strong performance on both tasks. Overall our experiments in this unconditional setting indicate that stochastic iterative target augmentation, at least in certain scenarios, is capable of leveraging the external property signal more effectively than an RL method.



```
def run():
  repeat(4):
    putMarker()
    move()
    turnLeft()
```

Figure 3.6: Illustration of our data generation process in the program synthesis setting. Given an input-output specification, we first use our generation model to generate candidate programs, and then select correct programs using our external filter. Images of input-output specification and the program A are from [16].

Program Synthesis Experiments

Finally, we present additional experiments using the conditional version of our method in the program synthesis domain, demonstrating its generalizability across domains. Program synthesis is the task of generating a program (using domain-specific language) based on given input-output specifications [16, 49, 31]. That is, the source is a set of input-output specifications for the program, and the target is a program that passes all test cases. Our method is suitable for this task because the target program is not unique. Multiple programs may be consistent with the given input-output specifications.

External Filter. The external filter is straightforward for this task: we simply check whether the generated output passes all test cases. Note that at evaluation time, each instance contains extra held-out input-output test cases; the program must pass these in addition to the given test cases to be considered correct. When we perform prediction time filtering, we do not use held-out test cases in our filter.

Experimental Setup. Our task is based on the educational Karel programming language [110] used for evaluation in [16] and [23]. Commands in the Karel language guide a robot's actions in a 2D grid, and may include for loops, while loops, and conditionals. Figure 3.6 contains an example. We follow the experiment setup of [16].

Evaluation Metrics. The evaluation metric is top-1 generalization. This metric measures how often the model can generate a program that passes the input-output test cases on the test set. At test time, we use our model to generate up to L candidate programs and select the first one to pass the input-output specifications (not including held-out test cases).

Models and Baselines. Our main baseline is the MLE baseline from [16]. This model consists of a CNN encoder for the input-output grids and an LSTM decoder along with a hand-coded syntax checker. It is trained to maximize the likelihood of the provided target program. Our model is the augmentation of this MLE baseline by our iterative target augmentation framework. As with molecular design, we generate up to $K = 4$ new targets per precursor during each augmentation step. Additionally, we compare against the best model from [16], which finetunes the same MLE architecture using an RL method with beam search to estimate gradients.[6] We use the same hyperparameters as the original MLE baseline; see [167] Appendix E.1 for details.

Results. Table 3.4 shows the performance of our model in comparison to previous work.

Our model (MLE+) outperforms the base MLE model in [16] model by a wide margin. Moreover, our model outperforms the best reinforcement learning model (RL + Beam Search) in

---

[6] More recently, [23] achieved state-of-the-art performance on the same Karel task, with top-1 generalization accuracy of 92%. They use a different architecture highly specialized for program synthesis as well as a specialized ensemble method. Thus their results are not directly comparable to our results in this paper for the MLE architecture.

Model	**Top-1**
MLE	71.91
MLE + RL + Beam Search	77.12
MLE+ (Ours)	**85.02**

Table 3.4: Model performance measured by top-1 generalization accuracy on Karel program synthesis task. MLE+ is our augmented version of the MLE model [16], while MLE + RL + Beam Search is their reinforcement learning method applied to the same architecture.

[16], which was trained to directly maximize the generalization metric. This demonstrates the efficacy of our approach in the program synthesis domain. Since our method is complementary to architectural improvements, we hypothesize that other techniques, such as execution based synthesis [23], can benefit from our approach as well.

Related Work

Molecular Design. Several previous works explore molecular design using different architectures. [132, 74, 46, 67] adopt generative modeling approaches for molecular design. [172, 114, 104] use reinforcement learning methods for this task. [63, 64] formulate this problem as graph-to-graph translation and significantly outperform previous methods in the conditional setting. However, their performance remains imperfect due to the limited size of given training sets.

On the other hand, recent advances in graph convolutional networks [33, 43] have provided effective solutions for the related problem of property prediction. Our work leverages strong property prediction models to improve the performance of generative models for molecular design, by checking whether generated molecules have desired chemical properties and augmenting the training set with molecules passing the property filter.

Program Synthesis. When correctness in program synthesis is defined by input-output test cases [16, 49, 31], one can check a generated program's correctness by simply executing it on each input and verifying its output. Indeed, [176, 23] use this idea in their respective decoding procedures, while also using structural constraints on valid programs. We leverage this ability to check correctness during training time data augmentation as well.

Reward-guided Generation. Recent work has proposed to incorporate rewards (e.g., properties) into generative models. In machine translation, [103] propose reward augmented maximum likelihood, which samples new targets from a *stationary* exponentiated payoff distribution centered at a ground truth target based on edit distance. Their approach is only viable when ground truth targets are given. In the case of molecular design, the number of ground truth targets is very limited. Our approach, based on stochastic EM, samples new targets from a learned non-stationary distribution which is not tied to any ground truth.

[62] use reinforcement learning to impose task-specific rewards for sequence generation, while [13] propose an adaptive sampling approach which generates additional targets based on parametric conditional density estimation. In contrast to these two approaches, our method is based on maximum likelihood and stochastic EM; [14] explore additional theoretical connections.

Semi-supervised Learning. Our method is related to various approaches to semi-supervised learning in different domains. In chemistry, [57] and [140] demonstrate pre-training approaches which use unlabeled molecules to learn initial representations for property prediction models. Our method instead tackles the problem of molecular generation, addressing the problem of limited data by generating additional data via a self-training technique. In machine translation, back-translation [133, 34] creates additional translation pairs by using a backward translation system to translate unlabeled sentences from a target language into a source language. In contrast, our method works in the forward direction because many translation tasks are not symmetric.

In image and text classification, data augmentation and label guessing [82, 10, 161] are commonly applied to obtain artificial labels for unlabeled data. Rather than generating new source-target pairs by augmenting the source side, we augment the target side. In syntactic parsing, our method is closely related to self-training [97]. They generate new parse trees from unlabeled sentences by applying an existing parser followed by a reranker, and then treat the resulting parse trees as new training targets. However, their method is not iterative, and their reranker is explicitly trained to operate over the top k outputs of the parser; in contrast, our filter is independent of the generative model. In addition we show that our approach, which can be viewed as iteratively combining reranking and self-training, is theoretically motivated and can improve the performance of highly complex neural models. Co-training [12] and tri-training [180, 20] also augment a parsing dataset by adding targets on which multiple baseline models agree. Instead of using multiple learners, our method uses task-specific constraints to select correct outputs.

Conclusion

In this work, we have presented a stochastic iterative target augmentation framework for molecular design. Our approach is theoretically motivated, and we demonstrate strong empirical results in both the conditional and unconditional molecular design settings, significantly outperforming baseline models in each case. Moreover, we find that stochastic iterative target augmentation is complementary to architectural improvements, and that its effect can be quite robust to the external filter's quality. Finally, in principle our approach is applicable to other domains as well.

3.2 Additional Projects

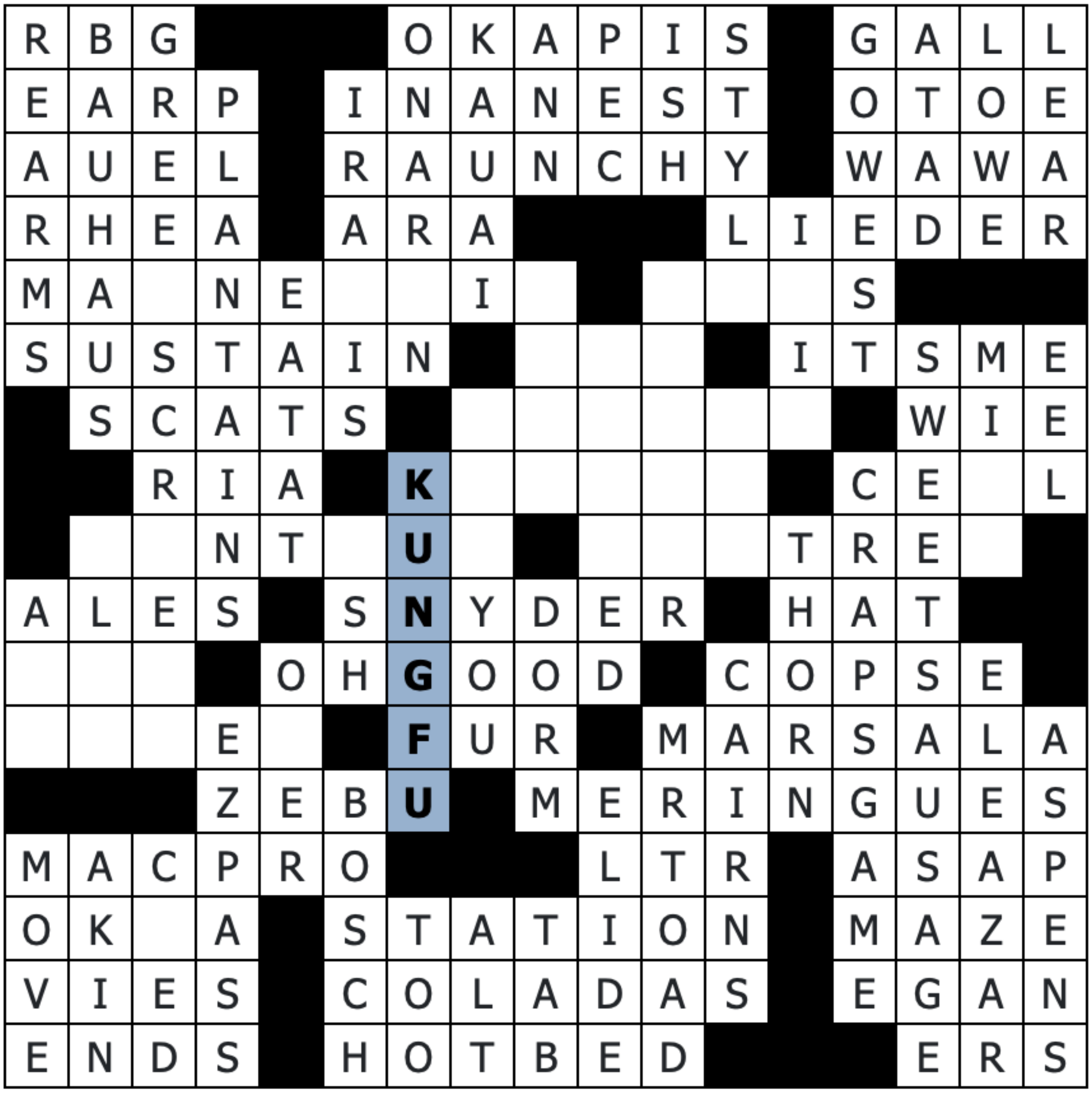
I will also briefly describe two other works in which I worked on control for structured domains.

Semantic Parsing

In [163] we introduce a novel setup for low-resource task-oriented semantic parsing which incorporates several constraints that may arise in real-world scenarios: (1) lack of similar datasets/models from a related domain, (2) inability to sample useful logical forms directly from a grammar, and (3) privacy requirements for unlabeled natural utterances. Our goal is to improve a low-resource semantic parser using utterances collected through user interactions. In this highly challenging but realistic setting, we investigate data augmentation approaches involving generating a set of structured canonical utterances corresponding to logical forms, before simulating corresponding natural language and filtering the resulting pairs. We find that such approaches are effective despite our restrictive setup: in a low-resource setting on the complex SMCalFlow calendaring dataset [3], we observe 33% relative improvement over a non-data-augmented baseline in top-1 match.

Crosswords

In [147] I collaborated on an automatic crossword solver, the Berkeley Crossword Solver. This solver uses Dense Passage Retrieval [68] to match questions to crossword answers, followed by solving a constrained decoding problem to fit answers into the crossword grid (see e.g., Figure 3.7, reproduced from [147]). Our solver uses loopy belief propagation followed by a local iterative improvement algorithm. Quantitatively, it solves 82% of New York Times crosswords perfectly, with up to 99.9% accuracy for individual letters. We ran our solver at the prestigious annual American Crossword Puzzle Tournament in 2021, where it outperformed all human solvers.



R	B	G	■	■	■	O	K	A	P	I	S	■	G	A	L	L
E	A	R	P	■	I	N	A	N	E	S	T	■	O	T	O	E
A	U	E	L	■	R	A	U	N	C	H	Y	■	W	A	W	A
R	H	E	A	■	A	R	A	■	■	■	L	I	E	D	E	R
M	A		N	E			I		■				S	■	■	■
S	U	S	T	A	I	N	■				■	I	T	S	M	E
■	S	C	A	T	S	■							■	W	I	E
■	■	R	I	A	■	**K**						■	C	E		L
■			N	T		**U**		■				T	R	E		■
A	L	E	S	■	S	**N**	Y	D	E	R	■	H	A	T	■	■
			■	O	H	**G**	O	O	D	■	C	O	P	S	E	■
			E		■	**F**	U	R	■	M	A	R	S	A	L	A
■	■	■	Z	E	B	**U**	■	M	E	R	I	N	G	U	E	S
M	A	C	P	R	O	■	■	■	L	T	R	■	A	S	A	P
O	K		A	■	S	T	A	T	I	O	N	■	M	A	Z	E
V	I	E	S	■	C	O	L	A	D	A	S	■	E	G	A	N
E	N	D	S	■	H	O	T	B	E	D	■	■	■	E	R	S

Figure 3.7: A partially-solved example crossword puzzle from the 2021 American Crossword Puzzle Tournament, where our system won first place against 1,100 top human solvers. The highlighted fill KUNGFU answers the wordplay clue: *Something done for kicks?*

Chapter 4

Long-Form Story Generation

Finally, I will discuss how I apply controllable generation methods to maintain quality over much longer model outputs, up to several thousands of words in length. I will focus on the domain of automatic story generation in this chapter, although many of the ideas are in principle applicable to other domains as well. This chapter will mainly cover two of my works, Re^3 [168] and DOC [166], with a brief discussion of other related projects I have worked on at the end.

4.1 Recursive Reprompting and Revision (Re^3)

Introduction

Generating long-term coherent stories is a longstanding challenge for artificial intelligence, requiring a comprehensive grasp of linguistic, world, and commonsense knowledge [21, 146]. Recently, many works have automatically generated short stories ranging in length from five sentences to one or two paragraphs [37, 171, 45, 123, 52]. While stories of such length serve as a good test bed for text generation, they are much shorter than typical short stories meant for human consumption, which are often several pages in length.

In this work, we aim to bridge some of this gap by generating much longer "short" stories: the final generated stories in our experiments are 2000-2500 words. We are the first to automatically generate plot-coherent stories of such length, with further length increases limited primarily by evaluation rather than technical issues.[1] Generating stories of such length faces qualitatively new challenges compared to prior work on shorter stories. First, the system must maintain a coherent overarching plot over thousands of words. Given an initial premise, it should maintain relevance to this premise over thousands of words as well. Additional challenges include preservation of narration style and avoiding factual contradictions over a very long horizon.

---

[1] We generate a 7500-word story in [168] Appendix M.

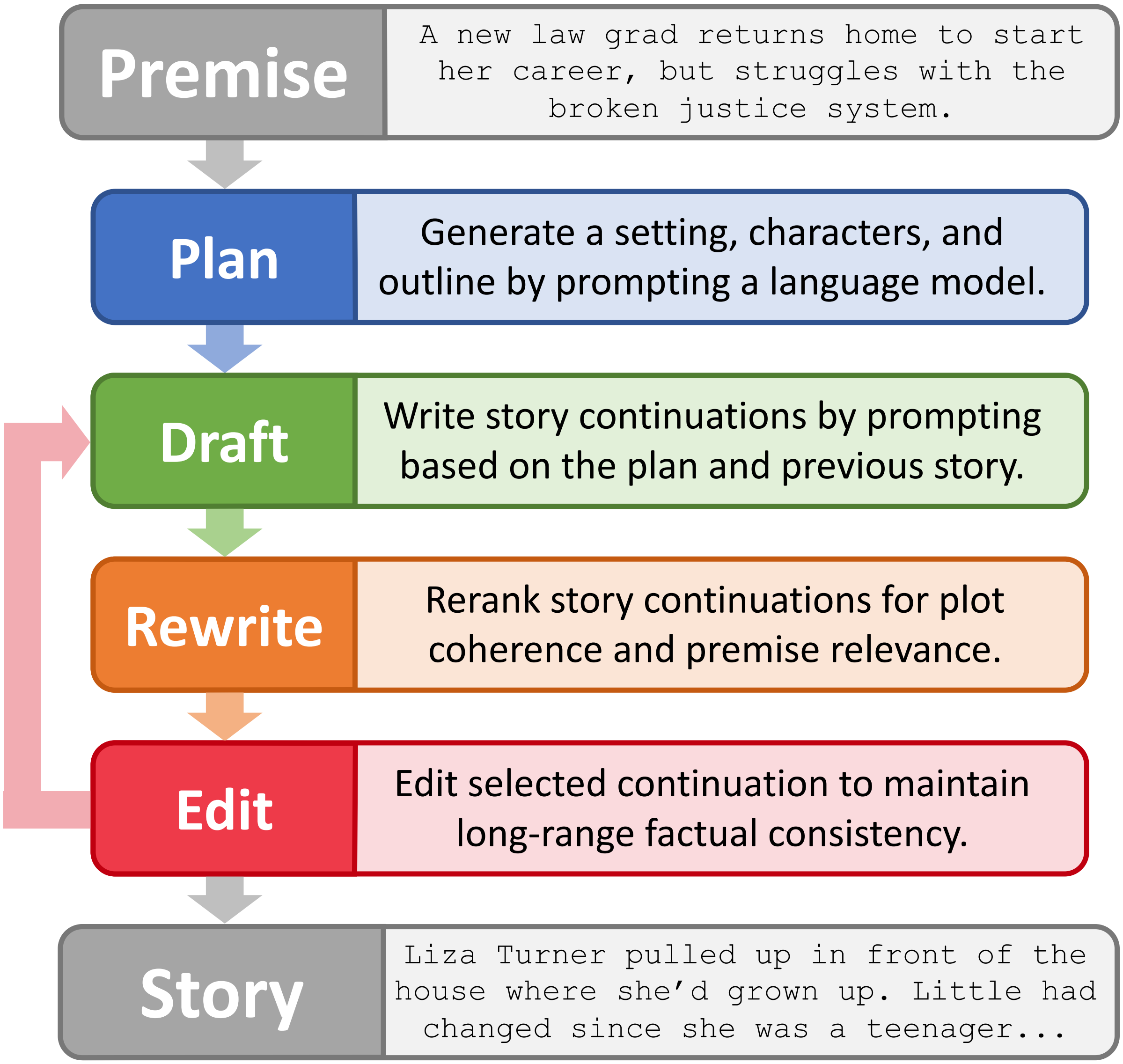

Premise
A new law grad returns home to start her career, but struggles with the broken justice system.
Plan
Generate a setting, characters, and outline by prompting a language model.
Draft
Write story continuations by prompting based on the plan and previous story.
Rewrite
Rerank story continuations for plot coherence and premise relevance.
Edit
Edit selected continuation to maintain long-range factual consistency.
Story
Liza Turner pulled up in front of the house where she'd grown up. Little had changed since she was a teenager...

Figure 4.1: High-level overview of Re^3.

Of course, recent years have also witnessed a dramatic rise in the capabilities of general-purpose (non-finetuned) large pretrained language models. Of particular note are their strong zero-shot capabilities, especially when given clever prompts [15, 72]. Yet despite recent improvements, even the best models to date may still struggle with complex long-form generation, such as in our story generation task.

In contrast, human writers successfully navigate the myriad challenges of long-form generation on a regular basis. We observe that a human writer does not simply write a long document in one shot. Rather, he or she may (a) create a detailed plan, then (b) draft each next passage of the document according to that plan. He or she may then revise by (c) rewriting passages entirely, and/or (d) post-editing for finer details.

Motivated by this observation, we propose the **Re**cursive **Re**prompting and **Re**vision framework (Re^3, Figure 4.1) to generate longer stories. While based on the human writing process, Re^3 is a fully automatic system with no human intervention, unlike prior approaches which model the human writing process with a human in the loop [44, 26, 83]. First, (a) Re^3's Plan module generates a plan by prompting GPT3 [15] to augment a given premise with a setting, characters, and outline. (b) Re^3's Draft module then generates each next story continuation by *recursively reprompting* GPT3 using a strategically crafted prompt, in a procedure which can be viewed as a generalization of chain-of-thought prompting [72]. Specifically, our prompt is dynamically reconstructed at each step by selectively manifesting contextually relevant information from the initial plan—itself generated by prompting—and the story thus far. We then divide the revision process into (c) a Rewrite module which emulates a full rewrite by reranking alternate continuations, and (d) an Edit module which makes smaller local edits to improve factual consistency with previous passages.

As an additional contribution, our Plan and Draft modules are fully zero-shot rather than trained on existing story datasets. Thus not only does Re^3 generate stories an order of magnitude longer than those of prior work, but it is not limited to any particular training domain.

To evaluate Re^3 for longer story generation, we compare its generated stories to similar-length stories from two GPT3-based "rolling-window" baselines. In pairwise comparisons, human evaluators rated stories from Re^3 as significantly and substantially more coherent in overarching plot (up to 14% absolute increase in the fraction deemed coherent), as well as relevant to the initial premise (up to 20%). In fact, evaluators predicted up to 83% of stories written by Re^3 to be written by humans. The results indicate that Re^3 can be highly effective at improving long-range coherence and premise relevance in longer story generation.[2]

Related Work

Automatic Story Generation. Several previous works have modeled parts of our proposed writing process, usually one part at a time.

Most similar to our Plan module are approaches using an outline or structured schema to maintain plot coherence [87, 37, 171, 45, 123, 143]. Other methods for high-level planning include latent variables [98, 154, 152], coarse-to-fine slot-filling [38], and keywords and/or control codes [112, 61, 162, 90].

Meanwhile, our Rewrite module uses rerankers similar to [48] and [153], although we model both coherence and premise relevance. [174] iteratively edits and improves the output like our Edit module, but we additionally *detect* when edits are required.

We emphasize again the length of stories we aim to generate. In prior studies, out-of-the-box language models struggled to generate even very short stories [54, 129]. Although there exist datasets of relatively longer stories, such as WritingPrompts [37] and STORIUM [2],

---

[2] All code and data available at `https://github.com/yangkevin2/emnlp22-re3-story-generation`.

many works still only focus on stories of about five sentences [154, 171, 116, 152], even when using language models with hundreds of billions of parameters [162]. Some challenges of generating longer stories are apparent in [152]: their method generates high-quality few-sentence stories, but their forced long text generations, while judged better than baselines', remain confusing and repetitive. Moreover, maintaining long-range plot coherence, premise relevance, and factual consistency is substantially harder over multiple-thousand-word horizons.

Human-In-The-Loop Story Generation. In contrast to fully automatic approaches like Re^3, several recent works have proposed human-interactive methods to maintain quality in longer stories [26, 83, 25]. Such works commonly combine both planning and revision systems [44, 26]. In principle, Re^3 is also highly controllable via human interaction, as both our planning and revision systems operate nearly entirely in natural language space; however, we focus on fully automatic generation in this work.

Prompting. Numerous works have demonstrated general-purpose language models' strong zero-shot ability on a wide variety of tasks via prompting [15, 179, 127, 108, 160]. Careful prompt design can yield further gains [81, 93, 72]. However, most prompting methods focus on shorter-answer tasks rather than long-form generation. Instead of generating the output in one shot, our recursive reprompting procedure treats prompting as a *subroutine* to generate the final output in conjunction with our planning and revision infrastructure. Compared to chain-of-thought prompting approaches like [72], Re^3 goes a step further by repeatedly re-composing the prompt in modular fashion, dynamically recombining the most contextually relevant parts of both the high-level plan and the story thus far.

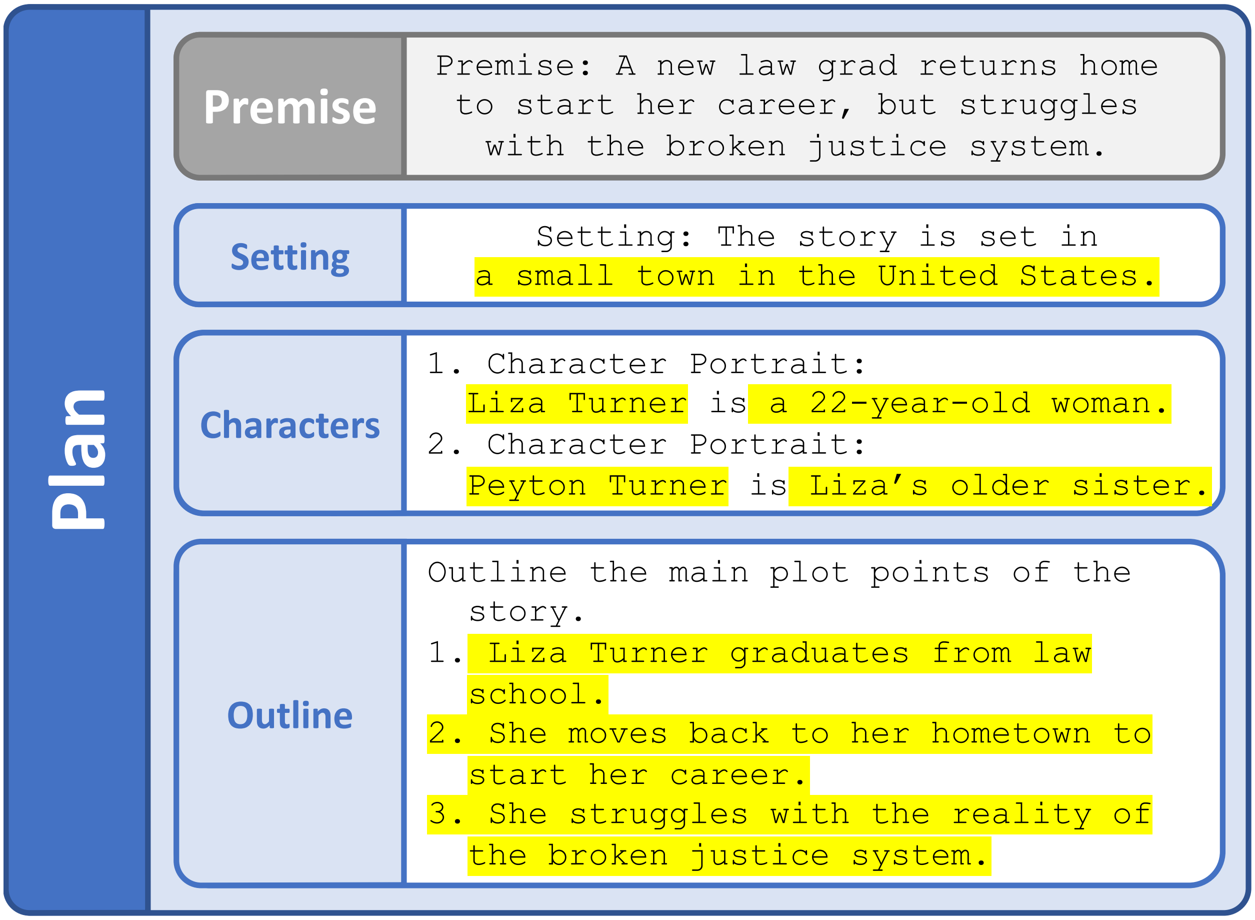
Recursive Reprompting and Revision

We now describe our Recursive Reprompting and Revision framework (Re^3), which decomposes the human writing process into our Plan, Draft, Rewrite, and Edit modules. See [168] Appendix K for concrete examples of each component in practice.

Plan Module

The Plan module augments a story premise with a setting, characters, and outline (Figure 4.2).

The setting is a simple one-sentence extension of the premise, obtained by using `The story is set in` to prompt GPT3-Instruct-175B [108], a version of GPT3 finetuned to better follow human instructions. Next, we use GPT3-Instruct-175B to generate up to three character names and then descriptions, conditioned on the premise and setting. For names, we do rejection sampling using simple heuristics to filter out malformed outputs ([168] Appendix A). Finally, we prompt GPT3-Instruct-175B to write a numbered outline of the story and parse the output into a list of outline points, re-sampling until the list is well-formed.



Premise: A new law grad returns home to start her career, but struggles with the broken justice system.

Setting: The story is set in a small town in the United States.

1. Character Portrait: Liza Turner is a 22-year-old woman.
2. Character Portrait: Peyton Turner is Liza's older sister.

Outline the main plot points of the story.
1. Liza Turner graduates from law school.
2. She moves back to her hometown to start her career.
3. She struggles with the reality of the broken justice system.

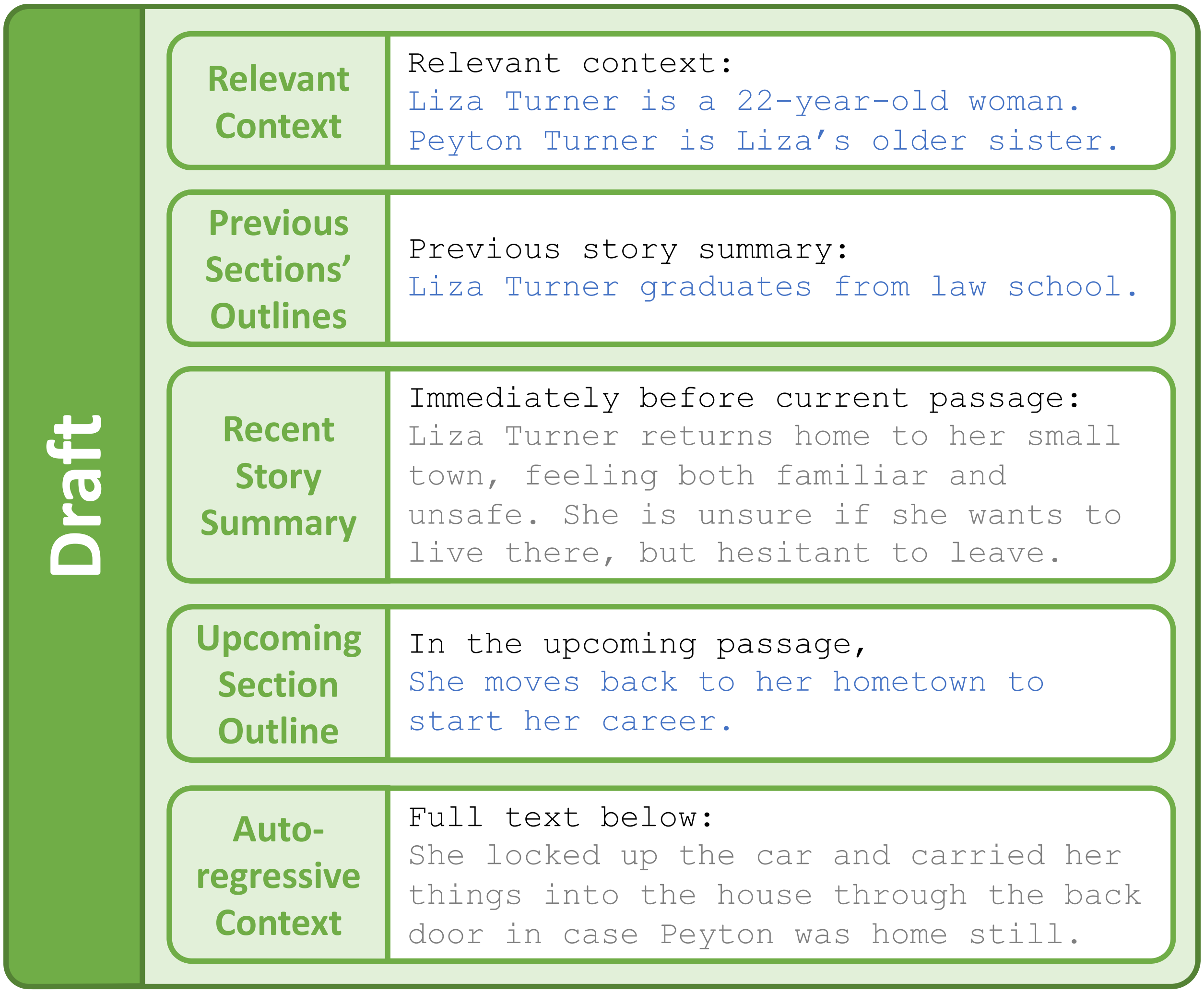
Figure 4.2: Illustration of Re^3's Plan module, which prompts a language model to generate a setting, characters, and outline based on the premise. Highlighting indicates generated text.

These plan components, themselves generated by prompting, will be repeatedly reused to compose prompts for generating story passages in the Draft module; hence *recursive reprompting*.

Draft Module

For each point of the outline, we will generate several story passages before moving on to the next outline point. Each passage is generated as a fixed-length continuation from a structured prompt, which is composed by our recursive reprompting procedure as shown in Figure 4.3.

The prompt begins with a selection of "Relevant Context" shown at the top of Figure 4.3. As the story progresses, we dynamically update the list of character descriptions using a named-entity-recognition-based pipeline, which identifies new entities from each new story passage using Flair [1] and writes descriptions using GPT3-Instruct-175B. Thus "Relevant



Relevant context:
Liza Turner is a 22-year-old woman.
Peyton Turner is Liza's older sister.

Previous story summary:
Liza Turner graduates from law school.

Immediately before current passage:
Liza Turner returns home to her small town, feeling both familiar and unsafe. She is unsure if she wants to live there, but hesitant to leave.

In the upcoming passage,
She moves back to her hometown to start her career.

Full text below:
She locked up the car and carried her things into the house through the back door in case Peyton was home still.

Figure 4.3: Illustration of the prompt constructed in Re^3's Draft module to generate each next story continuation. Our recursive reprompting approach combines pieces of the plan (blue) and previously generated story (grey) into a single prompt by concatenating the depicted components in order.

Context" initially contains all of the premise, setting, and characters shown in Figure 4.2, but subsequently selects only what is most relevant to the most recent story passage using a pretrained Dense Passage Retrieval (DPR) model [68].

The remainder of the prompt can be viewed as a coarse-to-fine description of the previous story, following the intuition that an author needs detailed information about the most recent passage but perhaps only higher-level information about much earlier passages. As shown in Figure 4.3, we include "Previous subsections' Outlines" as a very high-level summary of previous larger story subsections, followed by a "Recent Story Summary" written by GPT3-Instruct-13B[3] of a few penultimate passages. At the end we repeat verbatim the immediately preceding passage as "Autoregressive Context" from which point the story should continue. Finally, to enforce relevance to the current outline point, we include the "Current subsection Outline" in the prompt just before "Autoregressive Context."

Finally, the full prompt is fed to GPT3-175B to generate the next story passage.[4]

Rewrite Module

The generator's first output continuation is often low-quality, even with the planning and recursive reprompting in the Plan and Draft modules. Humans may encounter a similar problem after a first draft, particularly upon receiving feedback from others, and be forced to rewrite a passage altogether. Our Rewrite module models this rewriting process by reranking Draft module outputs based on coherence with the previous passage and relevance to the current outline point (Figure 4.4).

We note that this Rewrite module is the only part of Re^3 which uses prior story data. All of the modules which actually *generate* text (Plan, Draft, and to some extent Edit) do not require prior data.

Coherence Reranker. We train a discriminative model to predict whether a continuation is coherent with the previous story. As data, we split stories from the WritingPrompts dataset [37] into passages up to 1000 tokens long, labeling the ending up to 200 tokens as the gold continuation. Inspired by the contrastive learning setup of [153] and [48], we obtain negative examples by replacing the gold continuation with a random other continuation from either the same story or a different one. We then finetune a pretrained Longformer-Base [9] to classify whether a continuation is the true continuation for a given passage.

Relevance Reranker. We train a relevance model with the same architecture as our coherence model to predict whether a continuation is relevant to the current outline point.

---

[3] As economical usage of large language models is becoming increasingly important [138], we use the 13B model where we observe it is not substantially worse.

[4] This step does *not* use GPT3-Instruct-175B, as we observed in preliminary experiments that an earlier version of GPT3-Instruct-175B would frequently repeat subsections of the prompt. Generators other than GPT3-175B are also possible in principle: for example, retrieval-augmented architectures like RAG [86] or architectures designed for long-range dependencies like S4 [47]. However, it is critical to use a sufficiently high-quality language model: even scaling down to GPT3-13B resulted in noticeably less coherent outputs in our preliminary experiments.

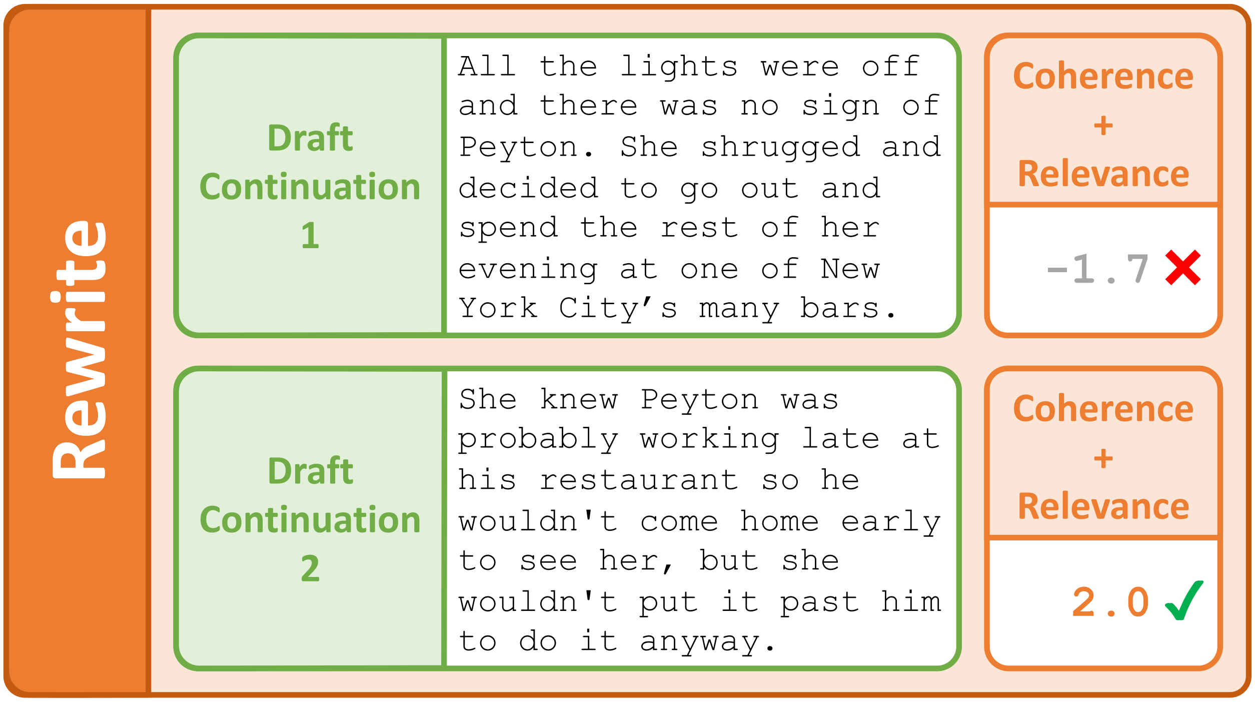

Rewrite
Draft Continuation 1
All the lights were off and there was no sign of Peyton. She shrugged and decided to go out and spend the rest of her evening at one of New York City's many bars.
Coherence + Relevance
-1.7 ✗
Draft Continuation 2
She knew Peyton was probably working late at his restaurant so he wouldn't come home early to see her, but she wouldn't put it past him to do it anyway.
Coherence + Relevance
2.0 ✓

Figure 4.4: Re^3's Rewrite module reranks the Draft module's continuations for coherence and relevance.

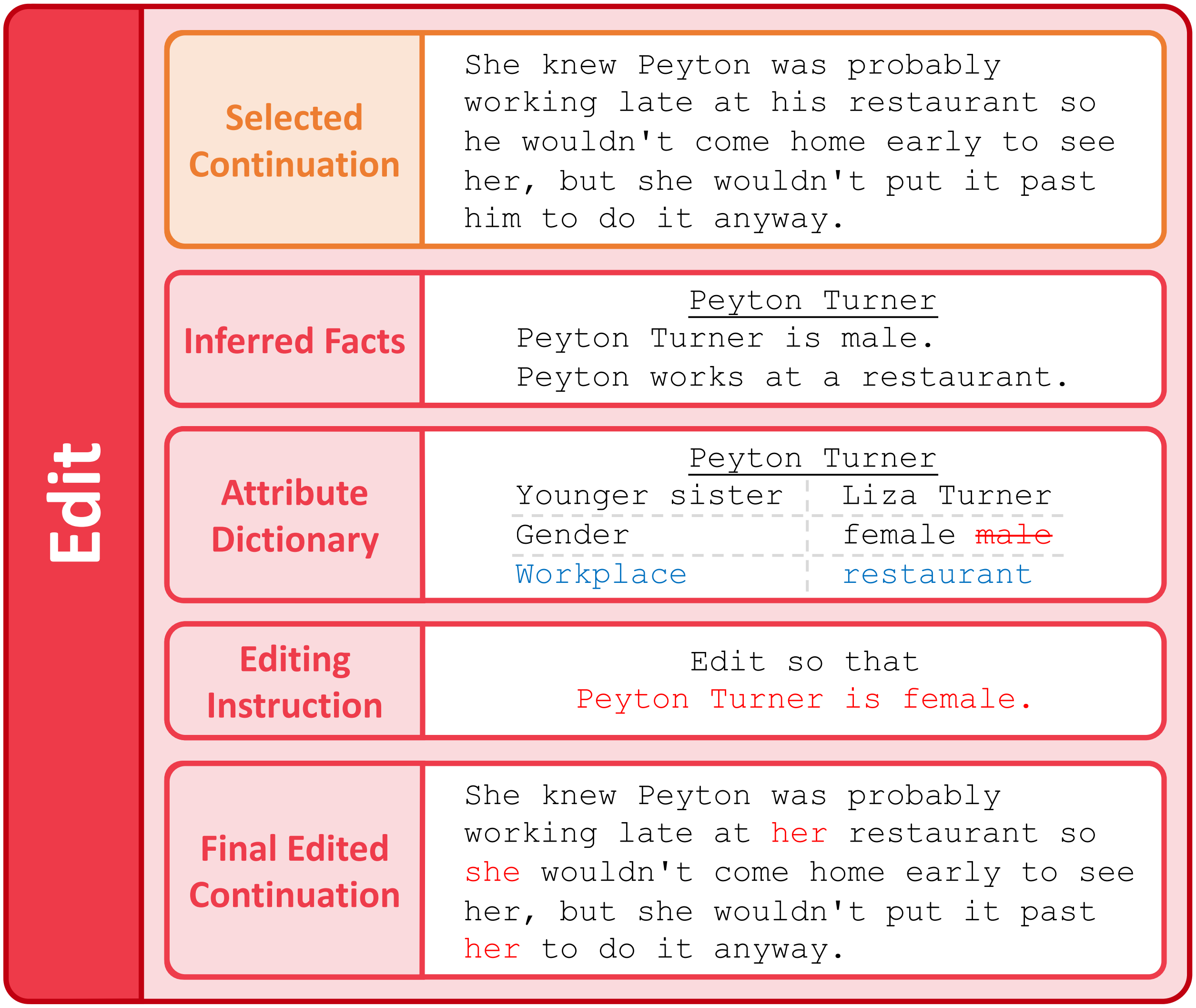
We construct a dataset of 2000 training examples, where each example consists of a 200-token story passage from WritingPrompts and a brief summary written by GPT3-Instruct-13B. Negative examples are constructed by selecting the summary of a different passage, whether in the same story or a different one.

Additional Heuristics. Finally, we filter out continuations with some writing problems which are easy to detect via rule-based heuristics. For example, we check for repetition issues, e.g., repeating chunks of the structured prompt. Similarly, to maintain consistent narration, we filter out first person continuations to enforce a consistent third person perspective. Full details in [168] Appendix B.

Edit Module

In contrast to the Rewrite module which reranks complete alternate continuations, the Edit module makes local edits to further refine a passage produced by careful planning, drafting, and rewriting.

Specifically, we aim to remove long-range factual inconsistencies. When a human detects a small factual discontinuity upon proofreading, he or she might simply edit the offending detail, rather than making major changes to the high-level plan or doing substantial rewrit-



She knew Peyton was probably working late at his restaurant so he wouldn't come home early to see her, but she wouldn't put it past him to do it anyway.

Peyton Turner

Peyton Turner is male.
Peyton works at a restaurant.

Peyton Turner

Younger sister	Liza Turner
Gender	female ~~male~~
Workplace	restaurant

Edit so that
Peyton Turner is female.

She knew Peyton was probably working late at her restaurant so she wouldn't come home early to see her, but she wouldn't put it past her to do it anyway.

Figure 4.5: Illustration of Re^3's Edit module. Starting from the Rewrite module's best continuation, we infer natural language facts about each character, and convert them to attribute-value pairs. New values (blue) are added to the attribute dictionary, and contradictory values (red) are corrected.

ing. Our Edit module mimics this process in two steps: *detecting* factual inconsistencies, and *correcting* them.

Detecting Factual Inconsistencies. An inconsistency involves two statements. As the number of statement pairs scales quadratically with story length, naively comparing all pairs can result in a sea of false positive "contradictions." Flagging inconsistencies while avoiding false positives requires overwhelming precision.

Task Framing. To make the task more tractable, we focus on factual inconsistencies in character attributes (e.g., age, occupation, relationship to another character). At a high level, our detection system maintains a compact knowledge base in the form of Figure 4.5's "Attribute Dictionary" for each character. With each new story passage, we check for contradictions against only these attribute-value dictionaries instead of all previous text. The dictionaries are then updated for the new passage, and new dictionaries are created for new characters when detected as described in our Edit module exposition.

Thus, the core of our detection system is a high-precision information extraction procedure for obtaining attribute-value pairs for a given character from a story passage. Rather than hard-coding a fixed set of attributes, our system is inspired by Open Information Extraction [35], in order to capture the wide variety of possible attributes which may be salient in different stories.

Implementation Details. We begin by prompting GPT3-Instruct-175B for a numbered list of facts about the given character, shown as "Inferred Facts" in Figure 4.5. Each fact is fed with a few-shot prompt to GPT3-Instruct-13B to extract attribute keys. We then prompt GPT3-Instruct-13B with the fact and each attribute key to obtain complete attribute-value pairs. In steps prone to hallucination, we generate three outputs and keep only those which are repeated, or entailed by other outputs according to a BART-Large-based [84] entailment model trained on MNLI [159]. See [168] Appendix C for complete details on information extraction, with example prompts.

Finally, we add new pairs to our dictionary, and use the entailment model to flag contradictions between new and old values for the same key.

Correcting Factual Inconsistencies. Once an inconsistency is detected, we frame the task of correcting it as controlled text editing. The original natural language fact (i.e., "Inferred Facts" in Figure 4.5) from which we extracted the contradicted attribute-value pair now becomes the basis for the "Editing Instruction" in Figure 4.5. This instruction is then fed along with the original continuation to the beta GPT3 Edit API.

Evaluation

Task Setup. We frame the task as generating a story given a brief initial premise. As a "story" is difficult to define in a rule-based manner, we do not impose any rule-based constraints on acceptable outputs, but will instead evaluate via several human-annotated metrics as described later.

Method	**Int.** ↑	**Coh.** ↑	**Rel.** ↑	**Hum.** ↑	**Misc. Prob.** ↓
ROLLING	45.0	45.7	44.0	74.0	1.20
RE^3	54.3	**60.0**	**64.0**	**83.3**	**1.07**
ROLLING-FT	52.7	48.7	49.3	74.7	1.48
RE^3	53.7	**60.0**	**65.3**	80.0	**1.35**

Table 4.1: Comparison of RE^3 against two baselines, ROLLING and ROLLING-FT, in two separate experiments examining comparisons of interestingness, coherence, relevance, humanlikeness, and counts of other problems. The first two rows show a pairwise comparison between ROLLING and RE^3 and the last two rows show the equivalent comparison between ROLLING-FT and RE^3. Bolding indicates significant differences with $p < 0.05$ on a paired t-test. Workers judged stories from RE^3 as significantly more coherent and relevant to the initial premise, in addition to having fewer writing problems.

To generate the initial premises, we prompt GPT3-Instruct-175B with high temperature to acquire 100 diverse premises.[5] All premises and stories are in English.

Method Instantiation. For fair comparison, it is desirable for the concrete implementation (henceforth RE^3) of our Re^3 framework to output stories of consistent length. While Re^3 is capable of generating shorter or longer stories (see e.g., our 7500-word example in [168] Appendix M), here we aim for roughly 3000 tokens (2000-2500 words).[6] Thus we re-sample the initial outlines until they contain exactly three points, and generate exactly four 256-token continuations for each outline point before moving on to the next. As a story-ending mechanism, we use the GPT3-175B Insert API to complete the story to the suffix "The End." Of course, more adaptive schemes for moving on to the next outline point and/or ending the story are possible, and we explore one possible "outline alignment" method in [168] Appendix M.

Baselines. As prior methods focus on dramatically shorter stories compared to Re^3, they are difficult to compare to directly.[7] Instead, we use the following two GPT3-175B-based baselines.[8]

1. ROLLING, a baseline which generates 256 tokens at a time via GPT3-175B using the premise and all previously generated story text as the prompt, left-truncating the

---

[5] Combining this simple premise generation scheme with Re^3 yields a story generation system which operates fully from scratch, with no input premise required.

[6] See [168] Appendix F for analysis on how story length may impact quality.

[7] Even the *premises* used as starting points in our task can be as long or longer than the final stories generated in several previous works. We believe that adapting any of the prior systems from our related work to function on our long-form story generation task could be an interesting contribution in its own right. In fact, Re^3 itself can be viewed as our attempt to extend and combine high-level planning and revision ideas from prior work, while simultaneously redesigning them to be able to leverage large out-of-the-box pretrained generators (GPT3), to scale up to long-form generation.

[8] Smaller (non-GPT3-175B) generators yielded qualitatively worse outputs in preliminary experiments.

prompt if it exceeds 768 tokens. Hence, a "rolling window" with maximum context length 1024 (the same maximum context length used in RE^3). After 3072 tokens are generated, we use the same story-ending mechanism as RE^3.
2. ROLLING-FT, which is identical to ROLLING except that GPT3-175B is first finetuned on several hundred passages from WritingPrompts stories of at least 3000 tokens.[9]

Metrics. As our main metrics, we track the percentage of stories which are:

1. **Interesting.** Interesting to the reader.
2. **Coherent.** Plot-coherent.
3. **Relevant.** Faithful to the initial premise.
4. **Humanlike.** Judged to be human-written.

We additionally track how often generated stories suffer from any of the following writing issues:

1. *Narration.* Jarring change(s) in narration and/or style.
2. *Inconsistent.* Factually inconsistent or containing very odd details.
3. *Confusing.* Confusing or difficult to follow.
4. *Repetitive.* Highly repetitive.
5. *Disfluent.* Frequent grammatical errors.

Binary indicators for these issues are summed and reported together as **Misc. Problems** in the main text, with individual numbers in [168] Appendix G.

All metrics are judged by Amazon Mechanical Turk workers, who are shown a premise and two corresponding stories in random order: one generated by RE^3 and one by a baseline. For interest, coherence, and relevance, workers indicate which story is better, or that both or neither are good. For the remaining metrics (humanlike and writing issues), workers mark each story individually. See [168] Appendix H for an example Mechanical Turk task.

Each pair of stories is judged by three workers. We report results individually for each pairwise comparison between RE^3 and each baseline, never mixing numbers from different comparisons. Workers are segregated by comparison to prevent bias.

Results. As shown in Table 4.1, RE^3 is highly effective at writing a longer story following a desired premise while maintaining a coherent overarching plot, validating our design choices inspired by the human writing process as well as our recursive reprompting approach to generation. RE^3 significantly and substantially improves over ROLLING and ROLLING-FT in both coherence and relevance. Annotators also marked RE^3's stories as having significantly fewer miscellaneous writing problems. Finally, RE^3 demonstrates strong performance in an absolute sense: annotators thought 83.3% and 80.0% respectively of RE^3's stories were written by humans in the two comparisons. Table 4.2 shows a heavily abridged example story by RE^3, exhibiting strong coherence and premise relevance.

---

[9] We initially considered a third rolling window baseline using GPT3-Instruct-175B rather than GPT3-175B, but observed that this baseline frequently devolved into highly repetitive text or gibberish. Thus we do not report a formal comparison. In any case, ROLLING is in some sense the best comparison, as RE^3 uses the same un-finetuned GPT3-175B generator.

PREMISE: After years of being content with only a small inheritance, Sarah's life is turned upside down when she discovers she is actually a millionaire. She must learn to navigate her new life of luxury while also dealing with the guilt she feels over her new-found fortune.

GENERATED OUTLINE:

1. Sarah learns that she is a millionaire.
2. Sarah must learn to navigate her new life of luxury.
3. Sarah deals with the guilt she feels over her new-found fortune.

GENERATED STORY:

Sarah Jensen never knew she had inherited a fortune until her mother died. In fact, she had never met her father ...[15 words]...

The only thing Sarah did know was that he had died long ago and that her mother had been left a sizeable inheritance. ...[23 words]...

Sarah's mother never spent any of the money ... all that mattered to her was how much it would mean to her daughter when she inherited it ...[81 words]...

Sarah learned that she was a millionaire. ...[417 words]...

She would never have had a chance like this - an opportunity to be financially independent, to live on her own ...[544 words]...

Now that she was rich, things were going to be different for her. She could afford a hotel suite and a lavish meal if she wanted it. ...[450 words]...

The words "Forgive me, father, for I have sinned" came unbidden into her mind as she reached out for her drink and took a large sip straight from the glass. ...[383 words]...

But the guilt was weighing on her now. ...[346 words]...

Table 4.2: A heavily abridged RE^3 example generation (see [168] Appendix J, Tables 39 and 40 for complete setup and story). While certainly not perfect, the story demonstrates strong long-range coherence and faithfulness to the premise.

Nonetheless, we observe qualitatively that RE^3 still has much room for improvement. Two common issues are illustrated in Table 4.3. First, while RE^3's stories almost always follow the premise to some degree—unlike our baselines' stories—they may fail to capture all parts of the premise, and may fail to follow parts of the outline generated by the Plan module (e.g., the first part of the premise and outline in Table 4.3). Second, due to failures in the Rewrite and especially Edit modules, there remain some confusing passages or contradictory statements: for example, in Table 4.3, the character Jaxon has a contradictory identity in some places.

However, unlike rolling window methods, RE^3's planning infrastructure is able to "self-correct" back to the original high-level plot despite early errors in generation. The latter part of the story in Table 4.3 illustrates this interesting capability.

See [168] Appendix J for additional complete, i.i.d. examples of stories from both RE^3

PREMISE: After the loss of her father, Shannon is determined to finish college and make something of herself. However, when she's sucked into a time portal and lands in the 19th century, Shannon has to find a way back to her own time before it's too late.

GENERATED OUTLINE:

1. Shannon is transported to the 19th century where she meets Jaxon Montgomery.
2. Shannon and Jaxon fall in love, but she is still determined to find a way back to her own time.
3. Shannon discovers that her father is still alive in the 19th century and she must choose between staying with Jaxon or returning to her own time.

GENERATED STORY:

Shannon was running late and hurried along the sidewalk. She was supposed to meet her best friend Hannah ...[423 words]...

When Jaxon proposed, we'd already been dating for over a year, so I [Hannah] went with it. ...[470 words]...

Thinking about him [Jaxon] made Shannon smile slightly as she recalled the first time they met ... right here in this backyard after noticing one another edging closer together ...[197 words]...

Shannon smiled when she looked into his eyes ...[176 words]...

[Jaxon asks,] "What do you mean that you're from the future?" ...[319 words]...

She looked him [Jaxon] straight in the eyes and said firmly, "I must go back to my time now....[199 words]...

She felt tears stinging in her eyes ...[73 words]...

There was no way she could ever go back to her own time after all this. ...[287 words]...

Shannon looked down at her feet again, then back up at Jaxon and said, "My father is alive?" ...[47 words]...

Jaxon gently rubbed Shannon's back in support and quietly said, "Yes, my love. He is alive and well. ...[52 words]...

Jaxon shook his head and said, "No, Shannon. I want you to be happy. And if that means going back to your own time, then so be it." ...[72 words]...

Shannon Randall vanished from the 19th century, never to be seen again.

Table 4.3: Another heavily abridged RE^3 example generation (see [168] Appendix J, Tables 24 and 25 for complete setup and story). RE^3 initially fails to follow the premise and outline, and in the beginning Jaxon is incorrectly introduced as Hannah's love interest. However, both issues are corrected in the subsequent story.

and baselines.

Method	**Int.** ↑	**Coh.** ↑	**Rel.** ↑	**Hum.** ↑	**Misc. Prob.** ↓
DRAFT-REWRITE-EDIT	50.3	46.7	50.7	70.0	1.33
RE^3	59.7	**63.3**	**63.7**	**80.0**	1.25
PLAN-DRAFT-EDIT	46.3	42.3	42.7	59.7	1.48
RE^3	**56.7**	**56.0**	**63.3**	67.3	**1.17**
PLAN-DRAFT-REWRITE	55.0	60.3	59.3	87.7	1.10
RE^3	57.0	57.3	59.3	87.0	1.12

Table 4.4: Ablations on individual components of RE^3, removing the Plan, Rewrite, and Edit modules respectively, examining comparisons of interestingness, coherence, relevance, humanlikeness, and counts of other problems. Each two rows show a pairwise comparison experiment between RE^3 and the corresponding ablation. Bolding indicates significant differences with $p < 0.05$. Both the Plan and Rewrite module are critical to performance, but the Edit module makes little difference.

Analysis

Ablation Study

Ablated Modules. We investigate the relative contribution of the individual modules of Re^3: Plan, Draft, Rewrite, and Edit. We ablate each module in turn as follows, except the Draft module as it is unclear how our system would operate without it.

1. DRAFT-REWRITE-EDIT, a version of RE^3 without the Plan module. Accordingly, we remove the recursive reprompting in Draft. Thus DRAFT-REWRITE-EDIT generates text identically to the ROLLING baseline, but is revised by our Rewrite and Edit modules.
2. PLAN-DRAFT-EDIT, a version of RE^3 without the Rewrite module reranking.
3. PLAN-DRAFT-REWRITE, a version of RE^3 which no longer edits using the Edit module.

Results. Table 4.10 shows that both the Plan and Rewrite modules, mimicking the human planning and rewriting processes, are critical for overall plot coherence and premise relevance. However, the Edit module contributes little to these metrics. We also observe qualitatively that there remain many continuity issues in RE^3's final stories which are not resolved by our Edit module, but which could be fixed by an attentive human editor. Such continuity issues range from non-character-centric inconsistencies, to facts which change over time, to outline plot points which were omitted in the story.

Further Analysis of Edit Module

We use a controlled setting to investigate if the Edit module can at least detect the character-based factual inconsistencies for which it is designed. We will refer to our detection subsystem as STRUCTURED-DETECT to avoid conflation with the Edit module as a whole.

Task Setup. We construct an evaluation dataset as follows. First we generate setups following our Plan module, up to but not including the outline. For each setup s we randomly resample a character's description until we manually observe a contradiction with the original, yielding a contradictory setup s'. For each of s and s', we generate a story (t and t'), resampling until the contradicted attribute appears in the story. If the resampling fails after 5 attempts we restart the whole procedure. We generate 50 (s, s', t, t') tuples in total; see [168] Appendix L for an example.

The task is then framed as classification: the method should judge (s, t) and (s', t') as consistent and (s, t') and (s', t') as contradictory. Thus the 50 (s, s', t, t') tuples yield 200 input pairs.

Baselines. We construct two simple baselines using the same BART-Large-MNLI entailment model used in STRUCTURED-DETECT. Given a (s, t) pair, the first baseline, ENTAILMENT, simply checks each sentence of s pairwise against each sentence of t, and returns the maximum probability of contradiction across all pairs. The second baseline, ENTAILMENT-DPR, checks each sentence of t against only one sentence of s based on relevance judged by DPR [68].

Results. As shown in Table 4.5, STRUCTURED-DETECT outperforms the two baselines according to the standard ROC-AUC metric for classification [53] when detecting character-based inconsistencies. Indeed, the most naive ENTAILMENT system's ROC-AUC score is barely better than chance performance (0.5), highlighting the core challenge wherein the detection system must be overwhelmingly precise. Moreover, STRUCTURED-DETECT is designed to scale to longer passages; we hypothesize that the performance gap compared to baselines would widen in an evaluation with longer inputs such as the stories from our main experiments.

Even so, the absolute performance of all systems remains low, even in this simplified setting. Additionally, many of our generated full stories contain non-character-based inconsistencies, such as in the setting or current scene. Some stories also contain false positives (flagged non-contradictions), such as character attributes which change over time.

Additionally, while we did not formally analyze the GPT3 Edit API's ability to *correct* inconsistencies after they are detected (as this system is largely not our contribution), we generally observed that it can fix isolated details but may struggle with larger changes. It also sometimes makes undesired edits or additions. Taken together, the compounding errors

Method	**ROC-AUC ↑**
ENTAILMENT	0.528
ENTAILMENT-DPR	0.610
STRUCTURED-DETECT	**0.684**

Table 4.5: ROC-AUC score of predicted contradiction probabilities for different methods on our evaluation set. STRUCTURED-DETECT outperforms our two entailment-based baselines.

from the detection and correction subsystems make it difficult for our current Edit module to effectively improve factual consistency over a multiple-thousand-word horizon, without simultaneously introducing unnecessary changes.

Discussion

We have considered the problem of automatically generating longer stories, proposing the Re^3 framework as an initial attempt at addressing the challenges of maintaining long-range coherence and premise relevance. Our RE^3 implementation exhibits strong performance on these metrics while generating stories over 2000 words long.

At its core, Re^3 is a system for emulating the human writing process for long-form generation while leveraging only general-purpose language models in the generation procedure. Thus concepts from Re^3 can potentially be adapted to non-story domains as well, especially the idea of dynamically re-injecting contextual information into a prompt. Moreover, should human interaction be desired, Re^3 is in principle highly controllable: most modules operate almost entirely in natural language.

Nonetheless, our main goal remains to further improve automatic long-form story generation. While RE^3's stories are an order of magnitude longer than those from prior work, most humans would still consider them to be "short stories"—and on the shorter side at that. Our long term goal is to generate interesting, long-range-coherent stories of greater length—perhaps what humans might call "novellas"—and eventually full-length novels. One step in this direction could be to extend Re^3 using multiple levels of hierarchical outline generation to obtain a much more detailed initial plan, as we do in [168] Appendix M to generate a 7500-word story.

In our view, the greatest barrier to further increasing story length is evaluation, which frustrates efforts to benchmark systems during both test time and development. In this work, we have compared RE^3 to baselines solely through human evaluation, which can be both noisy as well as costly even with non-expert annotators. While prior works have proposed some possible measures [8, 17], we hope that analyzing our generated stories (both RE^3 and baselines) can inspire further research on metrics for which we currently rely solely on human annotation. For example, while there exist reasonable metrics for text similarity on a sentence or paragraph level, long-form generation could benefit from metrics detecting when a longer passage begins on-topic but slowly veers off-topic, or when a passage uses on-topic vocabulary but is otherwise nonsensical in context. Similarly, improved metrics for *long-range* factual contradictions could greatly aid efforts to improve generations' factual consistency, such as our Edit module. Even if new metrics do not completely replace human annotations, they could help us both to evaluate longer stories as well as conduct more detailed ablation studies with larger sample sizes.

Additionally, while RE^3's stories are relatively plot-coherent and faithful to the premise, substantial gaps remain along other axes compared to even beginner human writers. One such axis is long-range factual continuity: while we believe our structured detection-correction method is a human-like approach, our current Edit module is certainly not human-level.

Moreover, human stories exhibit long-range continuity along many axes other than just factual attributes of characters, such as overall theme; scenes and world setting; pace and tempo of storylines; and foreshadowing before major events. It remains highly nontrivial to incorporate such considerations into automatic story generation.

4.2 Detailed Outline Control (DOC)

We next present an improved version of Re^3 which conducts more detailed, hierarchical planning and includes better control mechanisms for ensuring the plan is followed faithfully in downstream story generation, featuring my previously developed method FUDGE [164].

Introduction

Recent advancements in natural language generation systems have fueled increased interest in *long-form* text generation, in which texts may span thousands of words or more. Compared to tasks with shorter outputs, long-form generation involves meaningfully different challenges. It is nontrivial to maintain overarching coherence, or even basic relevance to an initial premise or plan. Even the most advanced language models to date, such as GPT4 [106], still cite long context as a major direction for further improvement, and require structured planning to generate text longer than a few hundred words.

In this work, we focus on long-form *story* generation, which is representative of the major difficulties in long text generation. Only recently have prior efforts even attempted to generate stories of comparable length to human-authored "short stories" (Re^3, [168]). Compared to humans, state-of-the-art story generation systems like Re^3 still fall short in numerous areas: common failure modes include insufficient high-level planning resulting in local fluency amid global incoherence, or deviating from said planning even when it exists.

To bridge some of this gap, we propose the Detailed Outline Control (DOC) framework. While reusing the high-level planning-drafting-revision structure of Re^3, DOC improves long-range plot coherence via two complementary approaches.

First, our *detailed outliner* refines a brief initial outline into a more detailed, hierarchical one (Figure 4.6 left). As motivation, a human author might also iteratively refine and expand a brief initial outline before drafting a long document, using the outline to guide a coherent plot rather than improvising plot points on the fly. Accordingly, our detailed outliner employs a structured prompting procedure to create a detailed outline with length scalable according to the desired scope of generation. Individual outline items are associated with a setting and characters, and are carefully filtered for relevance and coherence in context.

Second, our *detailed controller* maintains faithfulness to our detailed outline by controlling passage generation based on corresponding outline items (Figure 4.6 right). Because our detailed outline imposes many overlapping soft constraints, the detailed controller must exert sufficient control strength to enforce them. The detailed controller must also accommodate flexible natural language inputs and be computationally efficient when generating with

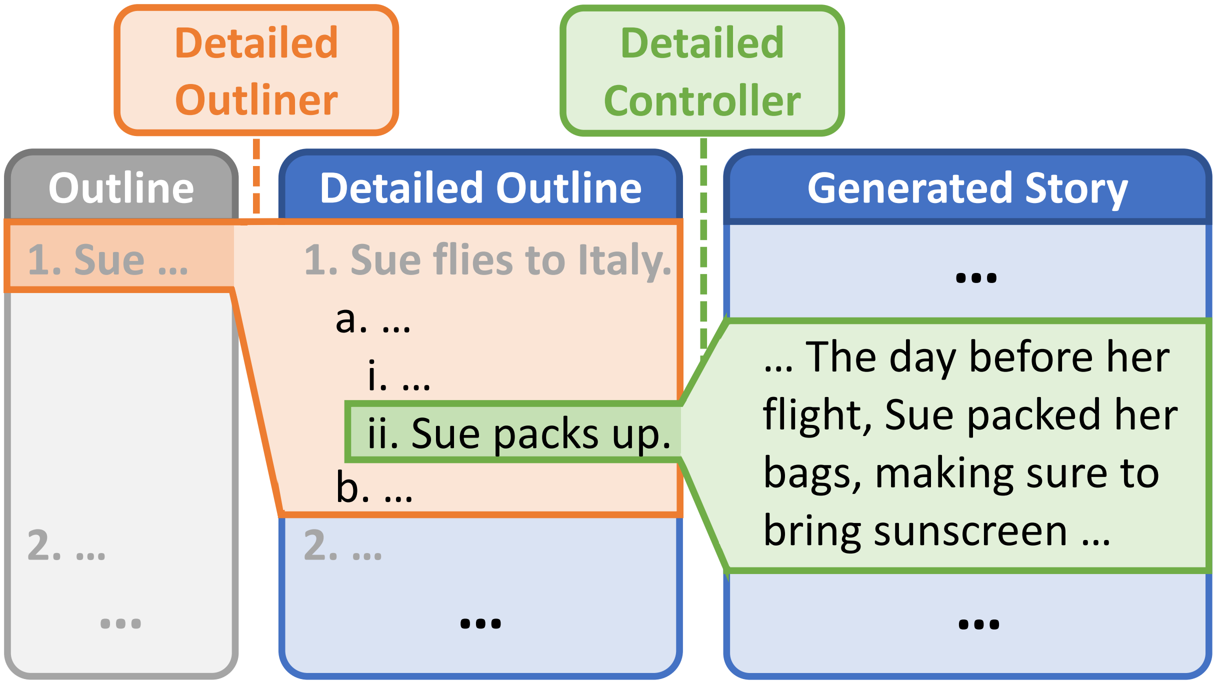


Figure 4.6: High-level overview of DOC. Our detailed outliner expands a brief initial outline into a more detailed outline. Our detailed controller then maintains faithfulness to the more detailed outline when drafting the main story.

state-of-the-art large language models. We implement the detailed controller as an OPT-350m-based controller according to FUDGE [164], designing a contrastive training procedure that aligns summaries to passage prefixes. In particular, we construct fluent hard negatives to encourage lengthy outputs to be not only initially on topic, but relevant throughout.

Compared to the original Re^3, the previous state of the art in long-form story generation, using DOC achieves dramatically higher plot coherence (22.5% absolute gain), outline relevance (28.2%), and even interestingness (20.7%) in pairwise human evaluations. Our ablations indicate that both the detailed outliner and detailed controller are critical. We also demonstrate that DOC can generate stories in collaboration with humans, interacting at a high-level planning stage rather than passage-by-passage as in many prior works [26, 83], and is overwhelmingly preferred over the original Re^3 in this setting.[10]

Related Work

Although we generate stories an order of magnitude longer compared to most prior works [154, 171, 116, 162, 152], we highlight below several works which employ related ideas.

---

[10] All code and models are available at `https://github.com/yangkevin2/doc-story-generation`.

Hierarchical Generation. A key component of DOC is our detailed outliner, which generates an outline hierarchically. Hierarchical structure in long-form generation can be implemented as part of the model architecture itself [170, 99, 50], or as natural language outlines or structured schema [**zhao-etal-2020-bridging**, **narayan-etal-2021-planning**, 37, 171, 45, 123, 143, 100, 168]. DOC's detailed outliner also builds a natural language outline, but can easily increase the level of detail to match the desired scope of the final story.

Controlled Generation. A second key component of DOC is the detailed controller, which increases faithfulness to our detailed outline. Prior works such as [56] use constrained decoding to guarantee rule-based constraints, while [29, 73, 164] propose modular control schemes based on an auxiliary model for a desired attribute. However, such methods typically do not handle natural language instructions.

In contrast, prompting [15, 179, 127, 160, 72, 108] offers a lightweight, flexible alternative. However, while prompts are an effective way to *provide context*, they may be insufficient for *enforcing constraints* due to the limited control strength, which is not easily tunable unlike in our detailed controller.

Human-In-The-Loop Story Generation. Some previous works generate longer stories with a human in the loop [44, 26, 83, 25, 60, 100]. We emphasize that DOC is designed to generate stories without human intervention. Nevertheless, due to planning in natural language space, DOC is in principle highly human-controllable. Unlike methods which interact with the human passage by passage [26, 83], DOC can also interact at a higher-level planning stage, as explored in our experiments.

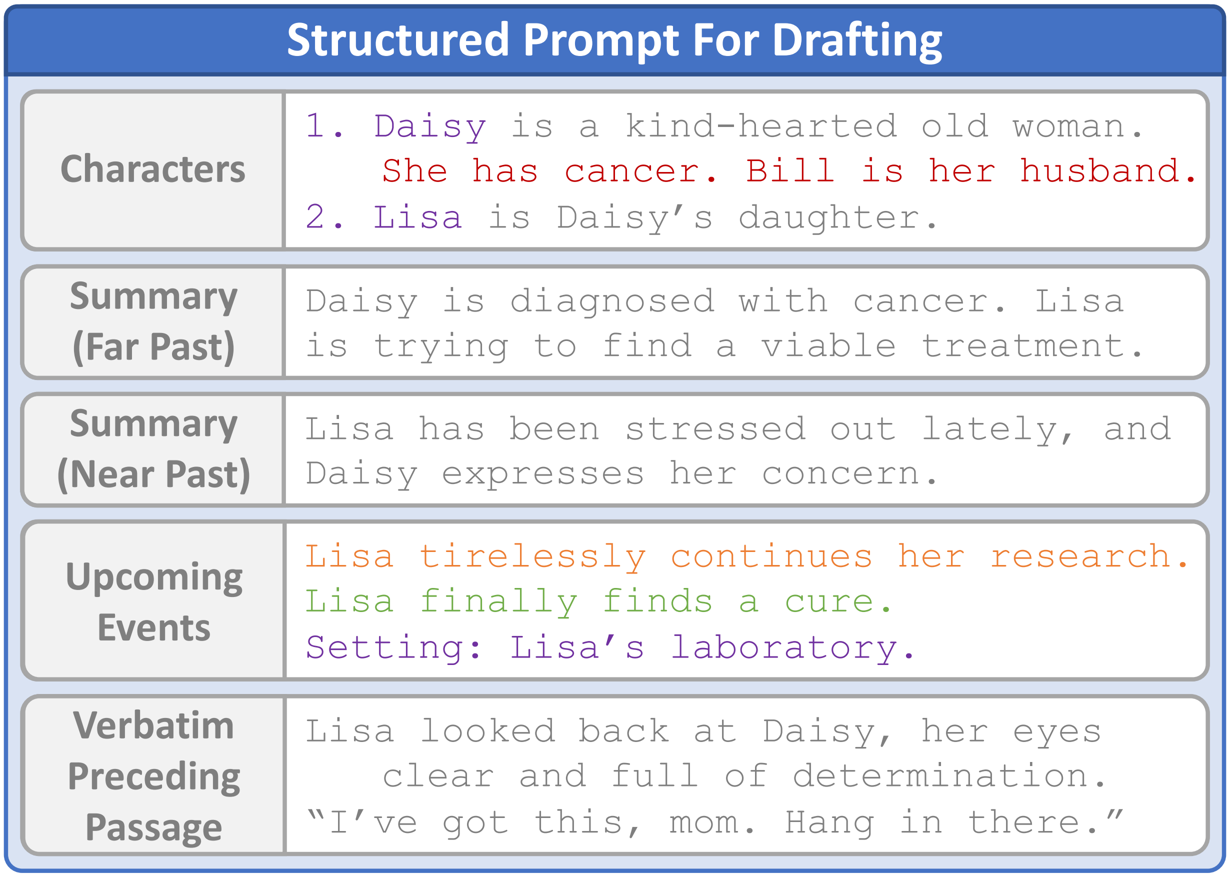
Detailed Outline Control

We introduce the Detailed Outline Control (DOC) framework, aiming to improve long-range plot coherence in automatically generated long stories.

Background and Motivation

A major inspiration for our work is Re^3 [168], which generates plot-coherent long-form stories of over 2000 words by decomposing the writing process into planning, drafting, rewriting, and editing steps. Their high-level plan contains a setting, character inventory, and brief three-point outline (e.g., Figure 4.6 "Outline"). In particular, when drafting each next story passage, they inject relevant context from the high-level plan and previously generated story via structured prompting (Figure 4.7). They finally rerank possible continuations using rerankers for outline relevance and passage coherence, and edit for consistency. DOC follows the high-level writing process and structured-prompting-based passage generation proposed by [168], though we remove the time-consuming editing step, which they find does not significantly affect final story quality.

However, [168] note that despite greatly outperforming simple rolling-window baselines, Re^3 still makes frequent errors in long-range coherence: some stories still contain lengthy



Structured Prompt For Drafting

Characters	1. Daisy is a kind-hearted old woman. She has cancer. Bill is her husband. 2. Lisa is Daisy's daughter.
Summary (Far Past)	Daisy is diagnosed with cancer. Lisa is trying to find a viable treatment.
Summary (Near Past)	Lisa has been stressed out lately, and Daisy expresses her concern.
Upcoming Events	Lisa tirelessly continues her research. Lisa finally finds a cure. Setting: Lisa's laboratory.
Verbatim Preceding Passage	Lisa looked back at Daisy, her eyes clear and full of determination. "I've got this, mom. Hang in there."

Figure 4.7: Stylized example showing the main components of the structured prompt used to draft new story passages in Re^3 and DOC. Leveraging our detailed outline and detailed controller, new elements of DOC's prompt include character development over time (red), more detailed events based on outline leaf nodes (orange), future context (green), and improved setting and character information (purple).

passages which seem not to fit the surrounding context, or deviate heavily from the initial outline. DOC aims to address these shortcomings via two major innovations: more detailed planning via our detailed outliner, and correspondingly fine-grained control during drafting via our detailed controller.

Detailed Outliner Motivation. While Re^3's outlines are plausible, they are insufficiently concrete, and do not scale to longer stories. A human author would not write a novel given just a three-sentence beginning, middle, and end. Not only can a more detailed outline empirically result in improved plot coherence, but it can enable greater control in human interaction as well, as we observe in our experiments.

Therefore, DOC constructs a detailed outline (e.g., Figure 4.6 "Detailed Outline") with depth adjustable according to the desired length of the final story. The detailed outline shifts creative burden from drafting to planning, reducing the need to improvise plot points on the fly during drafting.

Detailed Controller Motivation. The greater level of detail in our outline makes it much harder to stay faithful to that outline. To work with large language models such as GPT3-175B during drafting, prior works such as Re^3 have relied on clever prompting together with rejection sampling or reranking. However, prompting and reranking approaches are limited in the strength of control they can exert over the model distribution, which is especially problematic for systems like Re^3 which rely on complex constraints and long context in a structured prompt. Indeed, [168] observe that many of Re^3's stories already omit parts of even their brief three-point outline—and DOC's outline will impose far more detailed constraints.

Therefore, we design DOC's detailed controller to more strongly enforce the complex natural language constraints set by the outline. Our detailed controller, an adaptation of FUDGE [164], will operate token-by-token throughout generation instead of relying on only an initial prompt or post-hoc rejection sampling.

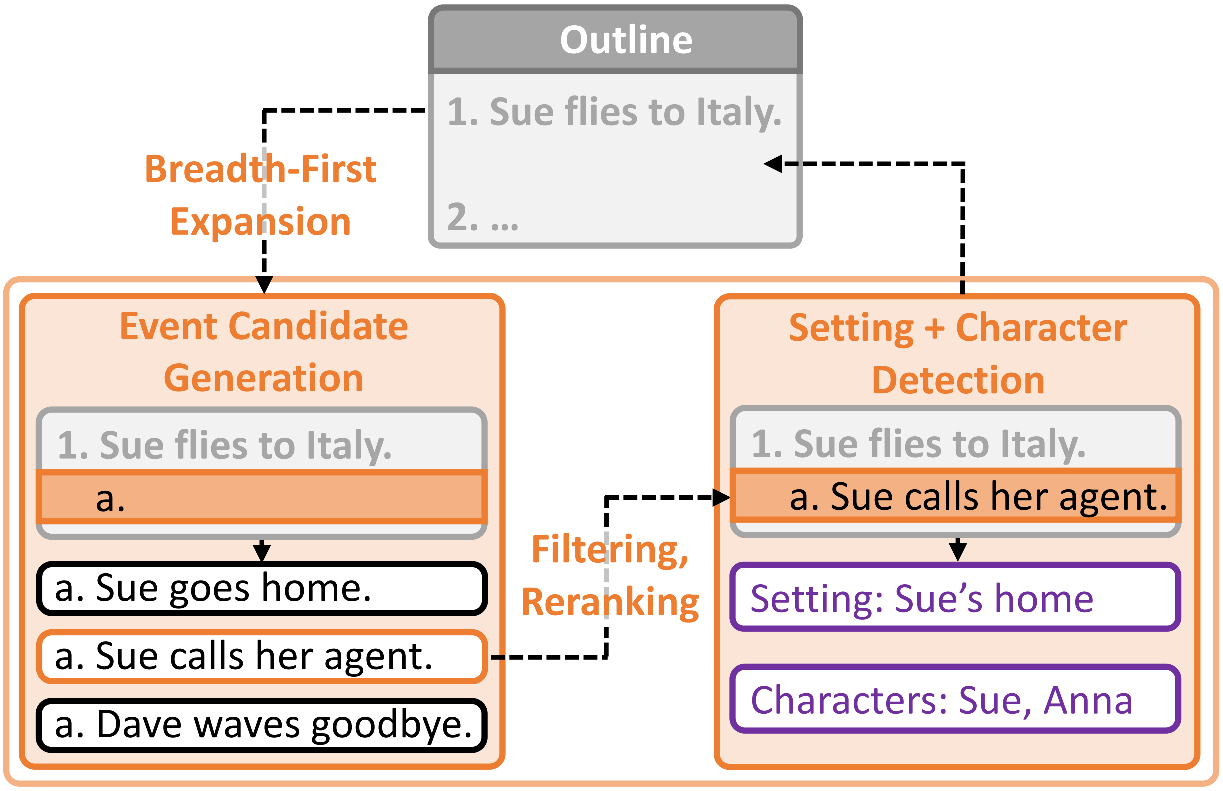


Figure 4.8: Diagram of new entry creation in the detailed outline. Our detailed outliner recursively expands outline items in breadth-first order. To create each new entry, it proposes candidate events and selects the best via filtering and reranking, and then detects the setting and relevant characters.

Detailed Outliner

Our detailed outliner recursively generates a hierarchical detailed outline at arbitrary granularity. Figure 4.8 summarizes the individual components.

Breadth-First Expansion. Viewing the outline as a tree $\mathbf{T}$ initialized as just a root node r, we generate children in breadth-first expansion order. Starting from the items of the initial top-level outline (depth 1), we generate all of their children (depth 2), then all childrens' children (depth 3), and so forth. For each parent node p, we generate children one by one, stopping when a child c's event description ends with the end-of-text token. We restart and resample for a given p if there are too few or too many children, although empirically this procedure almost always results in just two or three children. We terminate outline generation after reaching a pre-specified depth.

Event Candidate Generation To generate possible event descriptions for a new child c (Figure 4.8 bottom left), we use a structured prompting approach. To maintain coherence with pre-existing nodes, the prompt contains context from all of c's ancestors, together with their respective children; in this way we provide relevant context whose length scales linearly with depth. Suffix context is injected via the GPT3 Insertion API using InstructGPT3-175B (`text-davinci-002`), the most advanced GPT model at the time of our experiments. See [166] Appendix B.1 for an example prompt.

Filtering and Reranking. After generating several event candidates for each c, we select the best via filtering and reranking. Specifically, we remove ill-formed candidates or those which are highly repetitive compared to nodes not in c's ancestors,[11] as determined by both word overlap and an entailment model [79].

For the first child of each parent, we select the remaining candidate most relevant to the parent by sentence similarity [124]. For other children, to avoid repetition and improve plot coherence, we select via an ordering model that predicts if an event occurs in the correct location relative to nearby context. The ordering model is trained by finetuning `roberta-large` [95] to detect out-of-order events in short outline-like stories. See [166] Appendix A for complete details on our filtering and reranking pipeline.

Setting and Character Detection We further augment our outline by explicitly representing settings and characters for each outline item (Figure 4.8 bottom right), thus shifting additional creative work from drafting to planning.

Our setting and character list are obtained by prompting InstructGPT3-175B ([166] Appendix B.2). Characters are matched against an initial character inventory similar to that of Re^3, though we generate more characters since our outline is more detailed.

Drafting With Detailed Outlines

After constructing our detailed outline, story drafting largely follows Re^3's structured prompting procedure based on injecting context from the plan and previous story (Figure

---

[11] However, since c is a sub-event of its ancestors, it is acceptable to repeat parts of ancestor texts. If no candidates remain after filtering, we accept p as a leaf node which is already sufficiently concrete and does not require further expansion.

4.7; [166] Appendix B.4). However, instead of generating a fixed-length passage for each top-level outline item as in Re^3, we generate a *variable-length* passage for each *leaf* of our tree-structured outline $\mathbf{T}$ (Figure 4.7, orange text), since different leaves may contain events at differing levels of concreteness. Specifically, we reuse the outline relevance and text coherence rerankers from Re^3's rewriting stage to detect when drafting is done for the current outline item, implementing early stopping based on a score threshold. We also generate fewer tokens than Re^3 before reconstructing the structured prompt, for finer-grained control.

In the prompt, we additionally highlight the current setting (Figure 4.7, bottom purple text), especially changes in setting. Characters (Figure 4.7, top purple text) are also retrieved from the outline. In contrast, Re^3 selects relevant characters for each passage on the fly during drafting, and does not track setting information, which can result in unexpected changes in story setting.

Character Development Over Time. Taking advantage of our detailed outline, we explore a simple method to make DOC aware of character development over time, which Re^3 struggled to handle. Concretely, we attempt to infer a new fact about each character whenever they appear in the outline ([166] Appendix B.3), filtering out facts already entailed by a previously inferred fact from an earlier outline item. When drafting a story passage corresponding to a given outline item, retrieved character descriptions in the prompt context contain all facts inferred up to that outline item (Figure 4.7, red text).

Detailed Controller

Next, our detailed controller enhances the generator's ability to maintain relevance to our detailed outline. We implement the detailed controller as a FUDGE [164] controller to guide passage generation according to a given summary. However, we will modify the FUDGE training procedure to improve performance on longer outputs.

Lightweight, Adjustable-Strength, Natural Language Control. FUDGE is a modular and lightweight control scheme that adds logits at each token of generation based on a future-aware discriminator for a desired attribute. Control strength can be increased by multiplying the added logits, but it is nontrivial to handle natural language instructions.

We adapt FUDGE to handle natural language instructions for the specific task of guiding passage generation according to a short description. We collect a dataset of passage-summary pairs by prompting InstructGPT3-13B to summarize story passages from the WritingPrompts dataset [37]; these summaries can then be viewed as outline events corresponding to the original passages. We train the FUDGE discriminator contrastively by finetuning OPT-350m to predict whether a *passage prefix* matches a given summary. In particular, we construct hard negatives by matching passages with summaries from elsewhere in the same story.

The result is a computationally lightweight detailed controller which can guide passage generation according to a short natural language description, with adjustable control strength.

Training to *Maintain* Relevance. In our training data, passages are either entirely correct or entirely wrong for a given summary—even for "hard" negatives from the same story—so the discriminator learns to predict high probabilities for any roughly aligned passage at test time. The resulting controller allows longer passages to quickly stray off topic after starting out on topic.

Thus we construct even harder training negatives. Given a positive passage-summary pair, we split the passage at a sentence boundary, and replace the text after the sentence boundary with text from another passage in the same story (beginning at a sentence boundary). We thus obtain grammatically-fluent corrupted passages which begin correctly for a given summary, but eventually stray off topic. Prefixes of such passages ending after the sentence boundary can then be given the negative label during training. Thus our detailed controller learns to *maintain* high relevance to the input description.

Using the same methodology, we also construct "harder positives" by mixing negative prefixes with positive completions, improving the controller's ability to get back on track should it go astray.

Drafting With Detailed Control

During drafting, we illustrate the flexibility of our detailed controller by controlling passages according to three different types of constraints imposed by our detailed outline, as follows.

1. *Event.* We feed the event description (Figure 4.7, orange text) verbatim to the controller.
2. *Setting.* If the setting changed from the previous outline item, we construct an input "summary" stating that the characters move to the new setting, using lower control strength compared to the event description.
3. *Character.* If a character appears who did not appear in the previous outline item, we construct an input "summary" stating as such, again using lower control strength.

Control Strength. In practice, we must balance control strength: too low strength risks deviating from the constraint, while too high strength risks narrowly-focused, repetitive generations which sacrifice creativity. We aim to strike this balance dynamically during drafting by using a control strength of 0 initially for each outline item, incrementing it with each subsequent drafting step, until satisfying our early stopping criteria for moving to the next outline item and resetting back to 0.

Future Context in Generation. Context from future parts of the outline can help generated passages transition better to subsequent story events. However, including future plot points in the prompt risks premature generation of future events in the absence of proper control, which we observed when trying to include such context in Re^3. Our detailed controller remedies this issue to some degree by controlling more strongly toward the current outline item. Therefore, when drafting for a given outline item, we include the next outline item as future context in the prompt (Figure 4.7, green text).

Evaluation

Experiment Setup. Our setup is similar to [168]. The input is just a brief (English) premise, typically 30-60 words, sampled from InstructGPT3-175B. The output is a complete story. We do not impose further rule-based constraints, as it is unclear how to define a "story," let alone a "good" story. Instead, quality will be judged via human-annotated metrics.

Metrics. To decrease noise, we compare 1000- to 1500-word passages corresponding to the same top-level outline item, rather than complete stories.

We use three main metrics, similar to those from [168] ([166] Appendix C), adapted for comparing passages instead of complete stories:

1. *Coherent.* Percentage of passages judged plot-coherent by human annotators.
2. *Relevant.* Percentage judged faithful to the corresponding outline item.
3. *Interesting.* Percentage judged interesting.

Annotators are shown two passages side-by-side ([166] Appendix K.1); for each metric we ask them to annotate which passage is better, or possibly both or neither. Thus all numbers are meaningful only relative to the method being compared against. Each pairwise comparison is labeled by three annotators.

We use Surge AI for annotation due to observing higher-quality results compared to Amazon Mechanical Turk. We find higher agreement compared to [168] ([166] Appendix I), likely due to Surge AI and our more focused annotation task.

Method Instantiation. We henceforth refer to the concrete instantiation of our DOC framework as DOC. In particular, we set outline depth to 3 and limit the branching factor to be between 2 and 5, resulting in stories averaging roughly 3500 words in length. We limit the model context window to 1024 tokens as in [168], so final stories are substantially longer than the visible context at any step. The base generator used during drafting is OPT-175B [177], due to the practical issue of requiring deeper model access than the GPT3 API supports (specifically, *efficient* token-level access to logits). See [166] Appendix D for further discussion, and [166] Appendix E for complete hyperparameters.

Baselines. We run two baselines.

1. RE^3: Our main baseline is based on Re^3 [168], the only previous system we are aware of that automatically generates stories of comparable length. For fair comparison, we modify Re^3 to also use OPT-175B during drafting. Hyperparameters are set to their paper values, except for the number of generation steps per outline item, which we increase slightly to match average story length with DOC. We reuse the setting, characters, and top-level outline from DOC for RE^3, as the planning differs only slightly up to here (DOC only uses more characters, and generates the outline item-by-item instead of in one shot).
2. ROLLING-OPT: A sanity check using OPT-175B with the same context window as DOC and RE^3. The prompt contains the premise and top-level outline item ([166] Appendix

F), followed by a rolling window on the previously-generated story as fits in the prompt. ROLLING-OPT generates the same length of text per outline item as RE^3.

Method	Coherent	Relevant	Interesting
RE^3	45.1	37.1	39.4
DOC	**67.6**	**65.3**	**60.1**
ROLLING-OPT	38.0	25.4	25.4
DOC	**80.8**	**78.9**	**69.5**

Table 4.6: Pairwise comparisons of DOC against baselines on passages corresponding to top-level outline items from 20 stories. Bold indicates significance with $p < 0.05$. DOC stories are rated substantially more plot-coherent, outline-relevant, and interesting compared to RE^3 and ROLLING-OPT.

Results. As shown in Table 4.6, DOC passages are judged dramatically more plot-coherent and outline-relevant compared to RE^3, not to mention the weak ROLLING-OPT. The results confirm our intuition that plot coherence and outline relevance should benefit from shifting creative work from planning to drafting, together with improved control. Perhaps surprisingly, annotators also judged DOC's passages to be significantly more interesting, which our subsequent ablations suggest is a result of our more detailed (and more eventful) outline.

Of course, qualitative inspection reveals room for improvement. While DOC usually does not deviate heavily from the top-level outline—unlike RE^3, which is sometimes almost completely off-topic—DOC often fails to follow lower-level parts of the detailed outline. Long-range factual consistency also remains a problem in both DOC and RE^3. Occasional errors in the detailed outline can be particularly damaging, resulting in cascading errors during drafting. Additionally, outline leaves in DOC are often inconsistent in level of detail: some remain too vague while others seem over-expanded. Moreover, the detected settings and characters at times seem incorrect or incomplete.

Table 4.8 shows a heavily abridged story written by DOC according to the (also heavily abridged) detailed outline in Table 4.7. See [166] Appendix M for complete, i.i.d. examples of DOC plans and stories.

Human-Interactive Story Generation

We additionally evaluate DOC compared to RE^3 in an interactive setting, focusing on human controllability. Unlike prior human-in-the-loop approaches which operate passage by passage [26, 83], we explore interaction at a higher-level planning stage, though in principle DOC can also support passage-level interaction.

Experiment Setup. The human writes a story premise, from which we generate an initial plan with only a top-level (depth-1) outline. The human then edits for up to 5 minutes. The resulting intermediate plan $\mathbf{P}$ is used in both DOC and RE^3, which subsequently diverge. For

PREMISE: A young woman is determined to never get married and live her life alone, but when she meets a man who seems perfect for her, she begins to rethink her decision.

GENERATED OUTLINE:

1. Jenna Adams meets Brian Johnson and immediately feels drawn to him.
    a. Jenna Adams meets Brian Johnson and feels an instant connection to him.
    b. The two of them start dating and Jenna Adams begins to fall in love with Brian Johnson.
2. Jenna Adams starts to think that maybe marriage isn't so bad after all when Brian Johnson seems like the perfect man for her.
    a. Jenna Adams starts to think that maybe marriage isn't so bad when Brian Johnson seems like the perfect man for her.
    b. After much soul searching, Jenna Adams decides that she wants to marry Brian Johnson.
3. However, when Brian Johnson's ex-girlfriend shows up and tries to win him back, Jenna Adams realizes that marriage isn't for her after all and that it's better to be alone than with someone who doesn't truly love you.
    a. Jenna Adams overhears a conversation between Brian Johnson and his ex-girlfriend, Teresa Campbell.
    b. Jenna Adams confronts Brian Johnson about the conversation and Brian Johnson confesses that he still has feelings for Teresa Campbell.
    c. Jenna Adams breaks up with Brian Johnson.
    d. Jenna Adams decides that it's better to be alone than with someone who doesn't truly love you.

Table 4.7: Example of a premise and heavily abridged DOC outline (settings, characters, and depth-3 items omitted; see [166] Appendix M, Table 28 for complete plan).

DOC, we extend $\mathbf{P}$ with depth-2 and then depth-3 outline items, with up to 5 more minutes of editing after generating each depth. For RE^3 the human simply edits $\mathbf{P}$ for up to 10 more minutes. Thus both methods are allotted 15 minutes of total editing. We then generate stories according to the final edited plans.

Metrics. We asked workers to label the following metrics specific to the interactive experience.

1. *Intent.* Which system's passage better followed their original intent as author.
2. *Control.* Which system's workflow they felt gave them more control.
3. *Intuition.* Which system was more helpful or intuitive to work with.
4. *Quality.* Which system they would choose to write another story, if prioritizing quality.

The intent metric is passage-level, while all others operate on the complete story level. Annotators label which system is better for each metric, or no preference ([166] Appendix

GENERATED STORY:
...[85 words]...
The first time Jenna saw him she stopped short in the middle of the aisle between bookshelves and looked up at him, her heart beating faster. ...[331 words]...
Jenna Adams wanted their relationship to go somewhere. ...[106 words]...
Maybe marriage wasn't so bad after all. ...[419 words]...
[Jenna:] I love you, Brian Johnson. I want to be with you forever. I want you to give me a ring and ask me to marry you. ...[811 words]...
[Jenna:] I still love you, but I just cannot trust your promises anymore. ...[222 words]...
[Jenna:] I overheard the conversations that you had with Teresa Campbell ...[122 words]...
[Brian:] I want you in my life forever. But I am confused about how I feel towards you and Teresa Campbell. ...[285 words]...
Jenna Adams then threw the ring into the fire pit that was in their backyard. She left Brian Johnson standing there in shock. ...[244 words]...

Table 4.8: A heavily abridged DOC story generated from the outline shown in Table 4.7 (see [166] Appendix M, Table 29 for complete story). Although some issues remain, the story has a coherent overarching plot which follows the outline.

K.2).

Method	Intent	Control	Intuition	Quality
RE^3	17.3	5.0	5.0	15.0
DOC	**80.0**	**80.0**	**80.0**	**75.0**

Table 4.9: Pairwise comparison of DOC vs. RE^3 on 20 human-interactive story generation runs. Humans judged faithfulness to authorial intent, control over generation, system intuitiveness, and story quality. Numbers indicate the percentage of responses in favor of each system, with "no preference" responses omitted. Bolding indicates significance with $p < 0.05$. DOC is preferred by a wide margin on all metrics.

Results. As shown in Table 4.9, humans overwhelmingly preferred DOC's interaction paradigm to RE^3 on all four of our human-interactive metrics: at least three-fourths indicated DOC as superior on each metric. In optional free-form comments ([166] Appendix J), reactions to overall story quality vary widely from disappointed to pleased, but clearly indicate that DOC's stories are more faithful to the plot outline and authors' original intentions. The results confirm that DOC's more detailed outline and improved control during drafting lead to humans judging DOC as more controllable and more faithful to authorial intent.

Analysis

Ablation Study

Ablated Components. To ablate the two main components of DOC, we modify DOC as follows:

1. DOC-NOOUTLINE, which generates only according to the top-level outline instead of the full detailed outline, using fixed passage length per outline item (instead of early stopping) and a fixed-strength detailed controller.
2. DOC-NOCONTROL, which is identical to DOC except the detailed controller is turned off.

We reuse the same coherence, relevance, and interestingness metrics from Table 4.6.

Method	Coherent	Relevant	Interesting
DOC-NOOUTLINE	61.8	41.2	57.8
DOC	73.5	**64.7**	66.7
DOC-NOCONTROL	62.7	52.0	58.8
DOC	70.6	**73.5**	50.0

Table 4.10: Pairwise comparisons of DOC vs. ablations without the detailed outliner and detailed controller, respectively, on passages from 10 stories. Bold indicates significance with $p < 0.05$. Although the results on plot-coherence and interestingness are inconclusive, both the detailed outliner and detailed controller are important for outline relevance.

Results. As shown in Table 4.10, compared to both ablations, DOC maintains significantly higher relevance to top-level outline items. Thus both the detailed outliner and detailed controller meaningfully contribute to our method's ability to follow the high-level outline. Although the gaps in plot coherence and interestingness are not statistically significant, the ablations suggest that DOC's gain in interestingness compared to prior work is mainly due to the more detailed outline; if anything, the detailed controller may slightly hurt interestingness. Indeed—perhaps unsurprisingly—we observe qualitatively that further increasing control strength yields increasingly narrowly-focused, repetitive outputs at the expense of creativity.

Detailed Relevance Evaluation

We now examine DOC's faithfulness to the outline at the leaves instead of at the top level. For each leaf-node outline item, we ask one annotator whether the event specified in the leaf occurred in either the corresponding passage or in the immediately preceding and following passages ([166] Appendix K.3). We do the same for DOC-NOCONTROL.

Method	**Detailed-Relevant**
DOC-NOCONTROL	37.8
DOC	**58.5**

Table 4.11: Percentage of short passages that are faithful to corresponding outline leaf nodes, ablating the detailed controller. Bold indicates significance with $p < 0.05$. The detailed controller greatly improves relevance to leaf nodes.

Results. Table 4.11 confirms that the detailed controller substantially improves DOC's ability to follow low-level outline details during drafting.

However, the overall numbers remain low, pointing to two issues. First, the outline leaf itself may be problematic: it may be unexpected in context, or overly vague. Second, the detailed controller may be unable to sufficiently steer the generation without further increasing control strength (which may sacrifice fluency). Thus, while DOC is substantially more faithful to the outline compared to baselines, a good deal of headroom remains.

Discussion

We have presented the DOC framework for improving long-range coherence in long-form story generation. DOC uses a detailed outliner to shift creative work from drafting to planning, and employs a detailed controller to maintain faithfulness to the detailed outline during drafting. Compared to the prior state-of-the-art, Re^3, DOC dramatically improves the plot-coherence, outline relevance, and even interestingness of generated stories according to human annotators.

4.3 Additional Projects

Finally, I will also briefly describe some recent and ongoing works which dive deeper into particular facets of long-form story generation.

Personalization

In [150], I collaborated on a project for improving personalization in the evaluation of stories. We construct two datasets based on movie plots and automatically generated story outlines, ask humans to provide detailed feedback on several plots or outlines, and track annotator IDs in order to investigate personalization. In particular, we propose to anonymize and summarize data where needed in order to avoid biases originating from data contamination issues in pretraining. We then fine-tune LLaMA-13B to get a personalized story evaluator, PerSE, which predicts an unseen test-time reviewer's evaluation of a story plot along different axes given just one to three examples of previous evaluations by the same reviewer. We find that PerSE performs quite well in practice at predicting human ratings given just a small number of review examples, for instance significantly beating GPT-4 in this setting.

Pacing

In [155], I collaborated on a project for improving pacing in automatically-generated story outlines (and hence downstream stories). We observe that automatically generated story outlines from e.g., DOC [166] are often wildly inconsistent in pacing, with some parts being too detailed while others are too vague in comparison. We construct a large dataset of story passages and summaries written in similar styles using a data augmentation approach based on using LLMs to summarize, and use this dataset to train an evaluator of the pairwise concreteness or vagueness of two story passages. We then use this evaluator to implement a "vaguest-first expansion" procedure in generating the hierarchical outline in the style of DOC, and also use the evaluator to evaluate and re-sample child nodes as required. We find that our final method, CONCOCT, significantly improves pacing on both story outlines and downstream generated stories according to human evaluators, without compromising other attributes of story quality such as interestingness, plot coherence, and premise relevance.

Factual Consistency

I am also currently involved in an ongoing collaboration attempting to build a structured framework for maintaining factual consistency within a story outline, which we hope to extend to downstream stories to improve factual consistency. Our method substantially extends the basic ideas from the Edit module in Re^3. When generating each new point of a story outline, we extract facts and decompose them into atomic statements. We additionally categorize them into pre-facts, post-facts, or static facts depending on their window of validity. As a result, our framework can distinguish between legitimate factual contradictions and redundancies as opposed to simple changes over time within a story. We are aiming to complete our empirical evaluations and submit the work to a conference in the coming months.

Improved Generation Efficiency

Finally, in [182] I collaborated on a project that makes the automatic story generation procedure of DOC much more efficient, by fine-tuning a single end-to-end language model (LLaMA) to generate outlines in the style of DOC outlines without requiring the many hierarchical API calls used in DOC. The resulting generator is much more efficient without sacrificing quality.

I have also recently released an updated version of my automatic story generation code at `https://github.com/facebookresearch/doc-storygen-v2`. This updated code takes advantage of recent language model releases such as LLaMA-2 [144] and GPT-4 [106] to dramatically simplify our pipeline, making it both faster and easier to build upon our work.

Chapter 5

Conclusion

In this thesis, I have presented several works on controlling LLM outputs in different settings. First, in Chapter 2, I presented FUDGE [164] and RLCD [169], general control methods for natural language outputs covering both pure inference-time control as well as fine-tuning based alignment. Next, in Chapter 3, I presented control methods for more structured domains, ranging from molecule design and program synthesis to semantic parsing and crosswords. Finally, I applied control methods to much longer outputs in my automatic story generation works Re^3 [168] and DOC [166], in which I endeavor to generate stories thousands of words in length while preserving long-range plot coherence.

I will finally conclude with a discussion of potential future directions, focusing on ongoing challenges in long-form outputs.

Other Text Domains. I have focused on creative stories in Chapter 4, but I believe many of the high-level ideas could be applicable to other long-form text generation settings, such as Wikipedia articles or movie scripts. Generation in such settings could potentially benefit from detailed planning via an outline, combined with additional control to maintain faithfulness to the initial plan. Of course, many of our specific prompts would require substantial modification to adapt to a new domain.

Improved Human Interaction. In Chapter 4 we experimented with DOC in a human-interactive setting, enabling the human to interact with DOC at a high-level planning stage, in contrast to previous works which operated at the drafting level [26, 83]. We are excited to continue exploring novel forms of human interaction and co-creativity that become possible as automated generation capabilities continue to improve.

Scaling to Longer Texts. While our stories (exceeding 3500 words on average) are lengthy by neural text generation standards, they remain relatively short by human authors' standards. We hope to eventually develop systems which can scale to full-length novels. We believe DOC makes an important contribution toward this ambitious goal by generating outlines with granularity scalable to story length, while also providing better control mechanisms to maintain faithfulness to the outline during drafting. However, there remain major barriers to high-quality longer generations, two of which we describe below.

Evaluation. While some recent works have suggested metrics for longer generations [17, 96], there is currently no substitute for human judgments for long-form outputs of the scale that we are concerned with, due to the sheer length of evaluated passages and complexity of our metrics. For example, it is unclear how one might automatically measure overarching plot coherence, or especially interestingness. However, automated metrics for relevance may be more tractable, especially as applied to our more fine-grained experiments on low-level outline items with shorter passages (subsection 4.2). To facilitate such efforts, we have open-sourced all annotations collected during our experiments, in hopes that they prove useful for developing improved metrics for long-form generation.

Another wrinkle of evaluation with open-ended text generation is that the evaluation can be highly subjective—even expert humans may disagree on the relative merits of different outputs. Hence it may be worthwhile to investigate personalization, which I have begun to do so in collaborations such as [150].

Long-Range Consistency. A second major problem is internal consistency over long passages, of which one major component is factual consistency. While more detailed outlines may help somewhat in this respect, outlining improvements may fundamentally struggle to help with consistency of low-level details which may not be salient enough to appear in the outline at all. DOC's stories occasionally contain glaring errors, e.g., inconsistent names or genders, and errors sometimes occur even during outlining, leading to cascading errors during drafting. Moreover, we have not yet mentioned non-factual aspects of long-range consistency besides overarching plot coherence. Such aspects include maintaining consistent story pacing, or literary devices such as foreshadowing, which are themselves interesting directions for exploration. While I have started to investigate topics like pacing in more detail in recent collaborations such as [155], and have some ongoing work on factual consistency, substantial work remains to be done.

Bibliography

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. “Contextual String Embeddings for Sequence Labeling”. In: COLING 2018, 27th International Conference on Computational Linguistics. 2018, pp. 1638–1649.

[2] Nader Akoury et al. “STORIUM: A Dataset and Evaluation Platform for Machine-in-the-Loop Story Generation”. In: the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020.

[3] Jacob Andreas et al. “Task-oriented dialogue as dataflow synthesis”. In: (2020).

[4] Anthropic. Introducing Claude. 2023. URL: `https://www.anthropic.com/index/introducing-claude`.

[5] Amanda Askell et al. “A general language assistant as a laboratory for alignment”. In: arXiv preprint arXiv:2112.00861 (2021).

[6] Yuntao Bai et al. “Constitutional AI: Harmlessness from AI Feedback”. In: arXiv preprint arXiv:2212.08073 (2022).

[7] Yuntao Bai et al. “Training a helpful and harmless assistant with reinforcement learning from human feedback”. In: arXiv preprint arXiv:2204.05862 (2022).

[8] Regina Barzilay and Mirella Lapata. “Modeling local coherence: An entity-based approach”. In: (2008).

[9] Iz Beltagy, Matthew E Peters, and Arman Cohan. “Longformer: The long-document transformer”. In: arXiv preprint arXiv:2004.05150 (2020).

[10] David Berthelot et al. “Mixmatch: A holistic approach to semi-supervised learning”. In: arXiv preprint arXiv:1905.02249 (2019).

[11] G Richard Bickerton et al. “Quantifying the chemical beauty of drugs”. In: Nature chemistry 4.2 (2012), p. 90.

[12] Avrim Blum and Tom Mitchell. “Combining labeled and unlabeled data with co-training”. In: Proceedings of the eleventh annual conference on Computational learning theory. Citeseer. 1998, pp. 92–100.

[13] David H Brookes, Hahnbeom Park, and Jennifer Listgarten. “Conditioning by adaptive sampling for robust design”. In: arXiv preprint arXiv:1901.10060 (2019).

[14] David H Brookes et al. "A view of Estimation of Distribution Algorithms through the lens of Expectation-Maximization". In: arXiv preprint arXiv:1905.10474 (2019).

[15] Tom Brown et al. "Language models are few-shot learners". In: (2020).

[16] Rudy Bunel et al. "Leveraging grammar and reinforcement learning for neural program synthesis". In: arXiv preprint arXiv:1805.04276 (2018).

[17] Louis Castricato et al. "Towards a Model-Theoretic View of Narratives". In: 2021.

[18] Gilles Celeux, Didier Chauveau, and Jean Diebolt. "Stochastic versions of the EM algorithm: an experimental study in the mixture case". In: Journal of statistical computation and simulation 55.4 (1996), pp. 287–314.

[19] Jonathan D Chang et al. "Learning to Generate Better Than Your LLM". In: arXiv preprint arXiv:2306.11816 (2023).

[20] Eugene Charniak et al. "Parsing as language modeling". In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016, pp. 2331–2336.

[21] Eugene Charniak. "Toward a model of children's story comprehension." PhD thesis. Massachusetts Institute of Technology, 1972.

[22] Shouyuan Chen et al. "Extending Context Window of Large Language Models via Positional Interpolation". In: arXiv preprint arXiv:2306.15595 (2023).

[23] Xinyun Chen, Chang Liu, and Dawn Song. "Execution-guided neural program synthesis". In: International Conference on Learning Representations (2019).

[24] Eunbi Choi et al. "Prompt Injection: Parameterization of Fixed Inputs". In: arXiv preprint arXiv:2206.11349 (2022).

[25] John Joon Young Chung et al. "TaleBrush: Sketching Stories with Generative Pretrained Language Models". In: CHI Conference on Human Factors in Computing Systems. 2022, pp. 1–19.

[26] Andy Coenen et al. "Wordcraft: a Human-AI Collaborative Editor for Story Writing". In: (2021).

[27] Reuben Cohn-Gordon, Noah Goodman, and Christopher Potts. "Pragmatically informative image captioning with character-level inference". In: NAACL (2018).

[28] Ning Dai et al. "Style transformer: Unpaired text style transfer without disentangled latent representation". In: arXiv preprint arXiv:1905.05621 (2019).

[29] Sumanth Dathathri et al. "Plug and play language models: A simple approach to controlled text generation". In: (2019).

[30] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: (2018).

[31] Jacob Devlin et al. "Neural program meta-induction". In: Advances in Neural Information Processing Systems. 2017, pp. 2080–2088.

[32] Yann Dubois et al. “Alpacafarm: A simulation framework for methods that learn from human feedback”. In: arXiv preprint arXiv:2305.14387 (2023).

[33] David K Duvenaud et al. “Convolutional Networks on Graphs for Learning Molecular Fingerprints”. In: Advances in Neural Information Processing Systems (2015), pp. 2224–2232.

[34] Sergey Edunov et al. “Understanding back-translation at scale”. In: arXiv preprint arXiv:1808.09381 (2018).

[35] Oren Etzioni et al. “Open information extraction from the web”. In: Communications of the ACM 51.12 (2008), pp. 68–74.

[36] Benjamin Eysenbach et al. “Contrastive learning as goal-conditioned reinforcement learning”. In: Advances in Neural Information Processing Systems (2022).

[37] Angela Fan, Mike Lewis, and Yann Dauphin. “Hierarchical neural story generation”. In: arXiv preprint arXiv:1805.04833 (2018).

[38] Angela Fan, Mike Lewis, and Yann Dauphin. “Strategies for structuring story generation”. In: arXiv preprint arXiv:1902.01109 (2019).

[39] Jessica Ficler and Yoav Goldberg. “Controlling linguistic style aspects in neural language generation”. In: arXiv preprint arXiv:1707.02633 (2017).

[40] Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. “Bottom-up abstractive summarization”. In: arXiv preprint arXiv:1808.10792 (2018).

[41] Marjan Ghazvininejad et al. “Generating topical poetry”. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016, pp. 1183–1191.

[42] Marjan Ghazvininejad et al. “Hafez: an interactive poetry generation system”. In: Proceedings of ACL 2017, System Demonstrations. 2017, pp. 43–48.

[43] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: ICML (2017).

[44] Seraphina Goldfarb-Tarrant et al. “Plan, write, and revise: an interactive system for open-domain story generation”. In: (2019).

[45] Seraphina Goldfarb-Tarrant et al. “Content planning for neural story generation with aristotelian rescoring”. In: arXiv preprint arXiv:2009.09870 (2020).

[46] Rafael Gómez-Bombarelli et al. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: ACS Central Science (2018). DOI: `10.1021/acscentsci.7b00572`.

[47] Albert Gu, Karan Goel, and Christopher Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. In: arXiv preprint arXiv:2111.00396 (2021).

[48] Jian Guan et al. “A knowledge-enhanced pretraining model for commonsense story generation”. In: Transactions of the Association for Computational Linguistics 8 (2020), pp. 93–108.

[49] Sumit Gulwani. “Automating string processing in spreadsheets using input-output examples”. In: ACM Sigplan Notices. Vol. 46. 1. ACM. 2011, pp. 317–330.

[50] Mandy Guo et al. “Longt5: Efficient text-to-text transformer for long sequences”. In: arXiv preprint arXiv:2112.07916 (2021).

[51] Michael Gutmann and Aapo Hyvärinen. “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models”. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. 2010, pp. 297–304.

[52] Rujun Han et al. “Go Back in Time: Generating Flashbacks in Stories with Event Temporal Prompts”. In: 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL). 2022.

[53] James A Hanley and Barbara J McNeil. “The meaning and use of the area under a receiver operating characteristic (ROC) curve.” In: Radiology 143.1 (1982), pp. 29–36.

[54] Ari Holtzman et al. “The curious case of neural text degeneration”. In: (2019).

[55] Ari Holtzman et al. “Learning to write with cooperative discriminators”. In: arXiv preprint arXiv:1805.06087 (2018).

[56] J Edward Hu et al. “Improved lexically constrained decoding for translation and monolingual rewriting”. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019, pp. 839–850.

[57] Weihua Hu et al. “Strategies for pre-training graph neural networks”. In: arXiv preprint arXiv:1905.12265 (2019).

[58] Zhiting Hu et al. “Toward controlled generation of text”. In: ICML (2017).

[59] Jiaxin Huang et al. “Large language models can self-improve”. In: arXiv preprint arXiv:2210.11610 (2022).

[60] Daphne Ippolito et al. “Creative Writing with an AI-Powered Writing Assistant: Perspectives from Professional Writers”. In: arXiv preprint arXiv:2211.05030 (2022).

[61] Daphne Ippolito et al. “Unsupervised hierarchical story infilling”. In: Proceedings of the First Workshop on Narrative Understanding. 2019, pp. 37–43.

[62] Natasha Jaques et al. “Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control”. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org. 2017, pp. 1645–1654.

[63] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. “Multi-resolution Autoregressive Graph-to-Graph Translation for Molecules”. In: arXiv preprint arXiv:1907.11223 (2019).

[64] Wengong Jin et al. "Learning multimodal graph-to-graph translation for molecular optimization". In: International Conference on Learning Representation (2019).

[65] Vineet John et al. "Disentangled representation learning for non-parallel text style transfer". In: arXiv preprint arXiv:1808.04339 (2018).

[66] Marcin Junczys-Dowmunt et al. "Marian: Fast neural machine translation in C++". In: arXiv preprint arXiv:1804.00344 (2018).

[67] Seokho Kang and Kyunghyun Cho. "Conditional molecular design with deep generative models". In: Journal of chemical information and modeling 59.1 (2018), pp. 43–52.

[68] Vladimir Karpukhin et al. "Dense passage retrieval for open-domain question answering". In: arXiv preprint arXiv:2004.04906 (2020).

[69] Nitish Shirish Keskar et al. "Ctrl: A conditional transformer language model for controllable generation". In: arXiv preprint arXiv:1909.05858 (2019).

[70] Yoon Kim and Alexander M Rush. "Sequence-level knowledge distillation". In: arXiv preprint arXiv:1606.07947 (2016).

[71] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: arXiv preprint arXiv:1312.6114 (2013).

[72] Takeshi Kojima et al. "Large Language Models are Zero-Shot Reasoners". In: arXiv preprint arXiv:2205.11916 (2022).

[73] Ben Krause et al. "Gedi: Generative discriminator guided sequence generation". In: arXiv preprint arXiv:2009.06367 (2020).

[74] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. "Grammar Variational Autoencoder". In: arXiv preprint arXiv:1703.01925 (2017).

[75] Guillaume Lample et al. "Multiple-attribute text rewriting". In: International Conference on Learning Representations. 2018.

[76] Guillaume Lample et al. "Phrase-based & neural unsupervised machine translation". In: arXiv preprint arXiv:1804.07755 (2018).

[77] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: International Conference on Machine Learning. PMLR. 2020, pp. 5639–5650.

[78] Michael Laskin et al. "Cic: Contrastive intrinsic control for unsupervised skill discovery". In: arXiv preprint arXiv:2202.00161 (2022).

[79] Moritz Laurer et al. Less annotating, more classifying–addressing the data scarcity issue of supervised machine learning with deep transfer learning and bert-nli. 2022.

[80] Angeliki Lazaridou, Anna Potapenko, and Olivier Tieleman. "Multi-agent Communication meets Natural Language: Synergies between Functional and Structural Language Learning". In: arXiv preprint arXiv:2005.07064 (2020).

[81] Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. "Dialogue state tracking with a language model using schema-driven prompting". In: arXiv preprint arXiv:2109.07506 (2021).

[82] Dong-Hyun Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: Workshop on challenges in representation learning, ICML. Vol. 3. 2013, p. 2.

[83] Mina Lee, Percy Liang, and Qian Yang. "CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities". In: arXiv preprint arXiv:2201.06796 (2022).

[84] Mike Lewis et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: CoRR abs/1910.13461 (2019). arXiv: `1910.13461`. URL: `http://arxiv.org/abs/1910.13461`.

[85] Mike Lewis et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: ACL (2020).

[86] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: Advances in Neural Information Processing Systems 33 (2020), pp. 9459–9474.

[87] Boyang Li et al. "Story generation with crowdsourced plot graphs". In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 27. 1. 2013, pp. 598–604.

[88] Jiwei Li et al. "A diversity-promoting objective function for neural conversation models". In: arXiv preprint arXiv:1510.03055 (2015).

[89] Lucy Li and David Bamman. "Gender and representation bias in GPT-3 generated stories". In: 2021.

[90] Zhiyu Lin and Mark O Riedl. "Plug-and-blend: a framework for plug-and-play controllable story generation with sketches". In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 17. 1. 2021, pp. 58–65.

[91] Guoqing Liu et al. "Return-based contrastive representation learning for reinforcement learning". In: arXiv preprint arXiv:2102.10960 (2021).

[92] Nelson F Liu et al. "Lost in the Middle: How Language Models Use Long Contexts". In: arXiv preprint arXiv:2307.03172 (2023).

[93] Pengfei Liu et al. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing". In: arXiv preprint arXiv:2107.13586 (2021).

[94] Ruibo Liu et al. "Data Boost: Text Data Augmentation through Reinforcement Learning Guided Conditional Generation". In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020, pp. 9031–9041.

[95] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: arXiv preprint arXiv:1907.11692 (2019).

[96] Shahbuland Matiana et al. “Cut the CARP: Fishing for zero-shot story evaluation”. In: arXiv preprint arXiv:2110.03111 (2021).

[97] David McClosky, Eugene Charniak, and Mark Johnson. “Effective self-training for parsing”. In: Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics. Association for Computational Linguistics. 2006, pp. 152–159.

[98] Yishu Miao and Phil Blunsom. “Language as a latent variable: Discrete generative models for sentence compression”. In: arXiv preprint arXiv:1609.07317 (2016).

[99] Lesly Miculicich et al. “Document-level neural machine translation with hierarchical attention networks”. In: arXiv preprint arXiv:1809.01576 (2018).

[100] Piotr Mirowski et al. “Co-Writing Screenplays and Theatre Scripts with Language Models: An Evaluation by Industry Professionals”. In: CHI Conference on Human Factors in Computing Systems (2023).

[101] MosaicML. Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs. 2023. URL: `https://www.mosaicml.com/blog/mpt-7b`.

[102] Nathan Ng et al. “Facebook FAIR’s WMT19 News Translation Task Submission”. In: arXiv preprint arXiv:1907.06616 (2019).

[103] Mohammad Norouzi et al. “Reward augmented maximum likelihood for neural structured prediction”. In: Advances In Neural Information Processing Systems. 2016, pp. 1723–1731.

[104] Marcus Olivecrona et al. “Molecular de-novo design through deep reinforcement learning”. In: Journal of cheminformatics 9.1 (2017), p. 48.

[105] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: arXiv preprint arXiv:1807.03748 (2018).

[106] OpenAI. GPT-4 Technical Report. 2023. arXiv: `2303.08774 [cs.CL]`.

[107] OpenAI. Introducing ChatGPT. 2022. URL: `https://www.openai.com/blog/chatgpt`.

[108] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: arXiv preprint arXiv:2203.02155 (2022).

[109] Kishore Papineni et al. “BLEU: a method for automatic evaluation of machine translation”. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. 2002, pp. 311–318.

[110] Richard E. Pattis. Karel the Robot: A Gentle Introduction to the Art of Programming. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1981. ISBN: 0471089281.

[111] Jonathan Pei, Kevin Yang, and Dan Klein. PREADD: Prefix-Adaptive Decoding for Controlled Text Generation. 2023.

[112] Nanyun Peng et al. "Towards controllable story generation". In: NAACL Story-NLP Workshop. 2018.

[113] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014, pp. 1532–1543.

[114] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. "Deep reinforcement learning for de novo drug design". In: Science advances 4.7 (2018), eaap7885.

[115] Matt Post et al. "Improved speech-to-text translation with the Fisher and Callhome Spanish–English speech translation corpus". In: Proc. IWSLT. 2013.

[116] Lianhui Qin et al. "Counterfactual story reasoning and generation". In: (2019).

[117] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[118] Alec Radford et al. Improving language understanding by generative pre-training. 2018.

[119] Rafael Rafailov et al. "Direct preference optimization: Your language model is secretly a reward model". In: arXiv preprint arXiv:2305.18290 (2023).

[120] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: Journal of Machine Learning Research 21.140 (2020), pp. 1–67. URL: `http://jmlr.org/papers/v21/20-074.html`.

[121] Nazneen Rajani et al. "Can foundation models label data like humans?" In: Hugging Face Blog (2023). https://huggingface.co/blog/llm-v-human-data.

[122] Sudha Rao and Joel Tetreault. "Dear sir or madam, may i introduce the gyafc dataset: Corpus, benchmarks and metrics for formality style transfer". In: arXiv preprint arXiv:1803.06535 (2018).

[123] Hannah Rashkin et al. "PlotMachines: Outline-conditioned generation with dynamic plot state tracking". In: arXiv preprint arXiv:2004.14967 (2020).

[124] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: arXiv preprint arXiv:1908.10084 (2019).

[125] David Rogers and Mathew Hahn. "Extended-Connectivity Fingerprints". In: J. Chem. Inf. Model. 50.5 (2010), pp. 742–754.

[126] Elizabeth Salesky, Matthias Sperber, and Alex Waibel. "Fluent translations from disfluent speech in end-to-end speech translation". In: arXiv preprint arXiv:1906.00556 (2019).

[127] Victor Sanh et al. "Multitask prompted training enables zero-shot task generalization". In: arXiv preprint arXiv:2110.08207 (2021).

[128] John Schulman et al. "Proximal policy optimization algorithms". In: arXiv preprint arXiv:1707.06347 (2017).

[129] Abigail See et al. “Do massively pretrained language models make better storytellers?” In: (2019).

[130] Abigail See, Peter J Liu, and Christopher D Manning. “Get to the point: Summarization with pointer-generator networks”. In: arXiv preprint arXiv:1704.04368 (2017).

[131] Abigail See et al. “What makes a good conversation? how controllable attributes affect human judgments”. In: arXiv preprint arXiv:1902.08654 (2019).

[132] Marwin HS Segler et al. “Generating focussed molecule libraries for drug discovery with recurrent neural networks”. In: arXiv preprint arXiv:1701.01329 (2017).

[133] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Improving neural machine translation models with monolingual data”. In: ACL (2016).

[134] Sheng Shen et al. “Pragmatically informative text generation”. In: arXiv preprint arXiv:1904.01301 (2019).

[135] Charlie Snell, Dan Klein, and Ruiqi Zhong. “Learning by distilling context”. In: arXiv preprint arXiv:2209.15189 (2022).

[136] Teague Sterling and John J Irwin. “ZINC 15–ligand discovery for everyone”. In: Journal of chemical information and modeling 55.11 (2015), pp. 2324–2337.

[137] Nisan Stiennon et al. “Learning to summarize with human feedback”. In: Advances in Neural Information Processing Systems 33 (2020), pp. 3008–3021.

[138] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and policy considerations for deep learning in NLP”. In: arXiv preprint arXiv:1906.02243 (2019).

[139] Nishant Subramani, Samuel Bowman, and Kyunghyun Cho. “Can Unconditional Language Models Recover Arbitrary Sentences?” In: Advances in Neural Information Processing Systems. 2019, pp. 15258–15268.

[140] Fan-Yun Sun, Jordan Hoffmann, and Jian Tang. “InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization”. In: arXiv preprint arXiv:1908.01000 (2019).

[141] Zhiqing Sun et al. “Principle-driven self-alignment of language models from scratch with minimal human supervision”. In: arXiv preprint arXiv:2305.03047 (2023).

[142] Yuandong Tian. “Understanding deep contrastive learning via coordinate-wise optimization”. In: NeurIPS 35 (2022), pp. 19511–19522.

[143] Yufei Tian and Nanyun Peng. “Zero-Shot Sonnet Generation with Discourse-Level Planning and Aesthetics Features”. In: 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL). 2022.

[144] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: arXiv preprint arXiv:2307.09288 (2023).

[145] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: arXiv preprint arXiv:2302.13971 (2023).

[146] Scott R Turner. “The creative process: A computer model of storytelling and creativity.” In: (1994).

[147] Eric Wallace et al. “Automated Crossword Solving”. In: 2022.

[148] Eric Wallace, Mitchell Stern, and Dawn Song. “Imitation Attacks and Defenses for Black-box Machine Translation Systems”. In: arXiv preprint arXiv:2004.15015 (2020).

[149] Eric Wallace et al. “Universal adversarial triggers for attacking and analyzing NLP”. In: arXiv preprint arXiv:1908.07125 (2019).

[150] Danqing Wang et al. “Learning Personalized Story Evaluation”. In: arXiv preprint arXiv:2310.03304 (2023).

[151] Qixin Wang et al. “Chinese song iambics generation with neural attention-based model”. In: arXiv preprint arXiv:1604.06274 (2016).

[152] Rose E Wang et al. “Language modeling via stochastic processes”. In: (2022).

[153] Su Wang, Greg Durrett, and Katrin Erk. “Narrative interpolation for generating and understanding stories”. In: arXiv preprint arXiv:2008.07466 (2020).

[154] Tianming Wang and Xiaojun Wan. “T-CVAE: Transformer-Based Conditioned Variational Autoencoder for Story Completion.” In: 2019.

[155] Yichen Wang et al. “Improving Pacing in Long-Form Story Planning”. In: EMNLP Findings (2023).

[156] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. “Neural network acceptability judgments”. In: Transactions of the Association for Computational Linguistics 7 (2019), pp. 625–641.

[157] Robert L Weide. “The CMU pronouncing dictionary”. In: URL: http://www. speech. cs. cmu. edu/cgibin/cmudict (1998).

[158] David Weininger. “SMILES, A Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules”. In: J. Chem. Inf. Model. 28.1 (1988), pp. 31–36.

[159] Adina Williams, Nikita Nangia, and Samuel Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: `http://aclweb.org/anthology/N18-1101`.

[160] Yuhuai Wu et al. “Autoformalization with large language models”. In: arXiv preprint arXiv:2205.12615 (2022).

[161] Qizhe Xie et al. “Unsupervised data augmentation”. In: NeurIPS (2020).

[162] Peng Xu et al. “MEGATRON-CNTRL: Controllable story generation with external knowledge using large-scale language models”. In: (2020).

[163] Kevin Yang et al. “Addressing Resource and Privacy Constraints in Semantic Parsing Through Data Augmentation”. In: 2022.

[164] Kevin Yang and Dan Klein. “FUDGE: Controlled text generation with future discriminators”. In: NAACL (2021).

[165] Kevin Yang et al. “Analyzing Learned Molecular Representations for Property Prediction”. In: Journal of chemical information and modeling (2019).

[166] Kevin Yang et al. “Doc: Improving long story coherence with detailed outline control”. In: ACL (2023).

[167] Kevin Yang et al. “Improving Molecular Design by Stochastic Iterative Target Augmentation”. In: ICML (2020).

[168] Kevin Yang et al. “Re3: Generating longer stories with recursive reprompting and revision”. In: EMNLP (2022).

[169] Kevin Yang et al. “Rlcd: Reinforcement learning from contrast distillation for language model alignment”. In: arXiv preprint arXiv:2307.12950 (2023).

[170] Zichao Yang et al. “Hierarchical attention networks for document classification”. In: Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies. 2016, pp. 1480–1489.

[171] Lili Yao et al. “Plan-and-write: Towards better automatic storytelling”. In: 2019.

[172] Jiaxuan You et al. “Graph convolutional policy network for goal-directed molecular graph generation”. In: Advances in Neural Information Processing Systems. 2018, pp. 6410–6421.

[173] Lantao Yu et al. “Seqgan: Sequence generative adversarial nets with policy gradient”. In: Thirty-first AAAI conference on artificial intelligence. 2017.

[174] Meng-Hsuan Yu et al. “Draft and edit: Automatic storytelling through multi-pass hierarchical conditional variational autoencoder”. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. 02. 2020, pp. 1741–1748.

[175] Manzil Zaheer et al. “Big bird: Transformers for longer sequences”. In: arXiv preprint arXiv:2007.14062 (2020).

[176] Lisa Zhang et al. “Leveraging Constraint Logic Programming for Neural Guided Program Synthesis”. In: (2018).

[177] Susan Zhang et al. “Opt: Open pre-trained transformer language models”. In: arXiv preprint arXiv:2205.01068 (2022).

[178] Xingxing Zhang and Mirella Lapata. “Chinese poetry generation with recurrent neural networks”. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014, pp. 670–680.

[179] Ruiqi Zhong et al. “Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections”. In: arXiv preprint arXiv:2104.04670 (2021).

[180] Zhi-Hua Zhou and Ming Li. “Tri-training: Exploiting unlabeled data using three classifiers”. In: IEEE Transactions on Knowledge & Data Engineering 11 (2005), pp. 1529–1541.

[181] Banghua Zhu, Jiantao Jiao, and Michael I Jordan. “Principled Reinforcement Learning with Human Feedback from Pairwise or K-wise Comparisons”. In: arXiv preprint arXiv:2301.11270 (2023).

[182] Hanlin Zhu et al. “End-to-end Story Plot Generator”. In: arXiv:2310.08796 (2023).

[183] Daniel M Ziegler et al. “Fine-tuning language models from human preferences”. In: arXiv preprint arXiv:1909.08593 (2019).