# Methods for Controlling and Evaluating Large Language Models for Creative Domains

Robert Kenneth Morain

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Dan Ventura, Chair
Tony Martinez
Nancy Fulda
Porter Jenkins

Department of Computer Science

Brigham Young University

ABSTRACT

Methods for Controlling and Evaluating
Large Language Models for Creative Domains

Robert Kenneth Morain
Department of Computer Science, BYU
Doctor of Philosophy

Abstract goes here.

Keywords: language model, controllablity, knowledge

# ACKNOWLEDGMENTS

For Emi, Rosie, and Forrest.

## Table of Contents

## List of Figures

# Chapter 1

## Introduction

Introduction goes here.

## Chapter 2

## Symbolic Semantic Memory in Transformer Language Models

I hereby confirm that the use of this article is compliant with all publishing agreements.

Robert Morain, Kenneth Vargas and Dan Ventura, "Symbolic Semantic Memory in Transformer Language Models," IEEE International Conference on Machine Learning and Applications, 2022, pp. 992-998, doi: 10.1109/ICMLA55696.2022.00166.

### Abstract

This paper demonstrates how transformer language models can be improved by giving them access to relevant structured data extracted from a knowledge base. The methods for doing so include identifying entities in a text corpus, sorting the entities using a novel attention-based approach, linking entities to a knowledge base, then extracting and filtering the knowledge to create a knowledge-augmented dataset. We evaluate these methods with the WikiText-103 corpus using standard language modeling objectives. These results show that even simple additional knowledge augmentation leads to a reduction in validation perplexity by 81.04%. These methods also significantly outperform common ways of improving language models such as increasing the model size or adding more data.

## 2.1 Introduction

Currently, transformer language models are the gold standard [1] for most language tasks. The strength of these models comes from their extensive pretraining on statistical language modeling tasks.

In recent years, many improvements have been made to transformer language models. These improvements include adding layers and parameters [8] as well as making changes to the model architecture [25]. There has also been work put into prompt engineering [135] to help guide the model to produce some desired output. The work presented here demonstrates a different approach aimed at improving a transformer language model's ability to access semantic information.

One problem with traditional self-supervised language modeling tasks is that semantic knowledge is only tangentially acquired. To gain more semantic knowledge, these models typically become larger and are trained with more data. It has been argued elsewhere that these models have no way of reasoning about the knowledge they have acquired and instead are "haphazardly stitching together sequences of linguistic forms...observed in...vast training data, according to probabilistic information about how they combine, but without any reference to meaning" [2]. This work designs simple experiments to demonstrate how language models can be improved by giving them access to additional structured data rather than strictly relying on statistics.

To proceed, we draw inspiration from research on semantic memory as a cognitive process [113]. Unlike transformer models, humans rely on their semantic and episodic memory to understand language and decide how to respond. While there are connectionist and symbolic models for semantic memory [36], this work loosely draws inspiration from the symbolic approach. Also, since transformers are connectionist models already, using a symbolic model of semantic memory allows us to create a connectionist-symbolic hybrid which has proven to be effective on certain symbolic tasks [55].

Another significant influence on this work comes from Daniel Kahneman's research on reasoning and decision making. Kahneman proposes a cognitive model which distinguishes between System 1 and System 2 thinking [38]. While System 1 is fast, instinctive, and emotional, System 2 is slower, more deliberative, and more logical. The base transformer model can loosely be compared to Kahneman's System 1 fast thinking, while knowledge base integration resembles the slower System 2. Similar to how the SOAR cognitive architecture attempts to replicate various cognitive processes, a transformer-knowledge base hybrid seems appropriate in this case [45].

The idea of combining connectionist and symbolic models has recently become more popular. Bosselut *et al.* introduced Commonsense Transformers (COMET) [6], a GPT-2 model trained on ATOMIC [94] and ConceptNet [101] to automatically construct knowledge graphs. Miller *et al.* introduces key-value memory networks which are trained on structured data from Wikipedia to improve performance on question answering tasks [61]. The CLEVR dataset [35] is a visual question answering dataset specifically designed to test a model's reasoning abilities by minimizing biases that models can exploit. This work shares the goal of improving a model's reasoning abilities through symbolic methods.

[1] The top 10 models on SuperGLUE's [121] leaderboard are all transformer-based models

The contributions of this paper include:

- Methods for augmenting language datasets with knowledge base data including attention-based entity selection.
- Modification to GPT-2's causal mask to attend to additional knowledge data.
- Demonstrating the effectiveness of knowledge augmented data on language modeling, which reduces perplexity by 81.04%.

## 2.2 Related Work

There is an extensive body of research on knowledge augmented language models from which this paper takes inspiration. REALM uses a knowledge retriever which allows a language model to attend to relevant documents from a large corpus such as Wikipedia [32]. While incorporating increasingly larger contexts into language models has proven to be a fruitful vein of research, Guu *et al.* claim to extend this progression beyond sentences and paragraphs all the way to the corpus level. We follow their lead by integrating knowledge from a large knowledge source. However, this work explores the potential of structured knowledge from a knowledge base rather than a document database.

KEPLER integrates knowledge into a language model by jointly training on language modeling and knowledge embedding objectives [122]. This causes the language model to build entity representations that contain an entity's description and are close to other linked entities in a knowledge base. Here, we utilize structured knowledge without requiring auxiliary objective functions.

KALM augments a recurrent neural network [52] by allowing the model to sample from vocabularies collected from a knowledge base and grouped by entity type. While KALM learns to identify entities in an unsupervised fashion, we follow the lead of models like KnowBert [82] which requires entity selection and linking. KnowBert implements a Knowledge Attention and Recontextualization mechanism which can be added to a layer in the language model to integrate knowledge from a knowledge base. This method combines pretrained entity embeddings with BERT-encoded token hidden states. Unlike KnowBert, our methods utilize a single model to encode and attend to the added knowledge.

## 2.3 Methods

These methods outline one way to augment a text corpus with semantic knowledge. While this process is general to any dataset or knowledge base, a description of the specific implementation details are provided.

### 2.3.1 Augmented Dataset Creation

The augmented dataset creation process is composed of a *knowledge base*, an *entity selection* algorithm, a *knowledge extraction* process, and a *merge* between the extracted knowledge and the original dataset. Figure 2.1 shows how these systems work together to select the supplementary data to include in the augmented dataset. For each sequence in the dataset, a set of entities is extracted from the sequence and filtered based on the entity selection

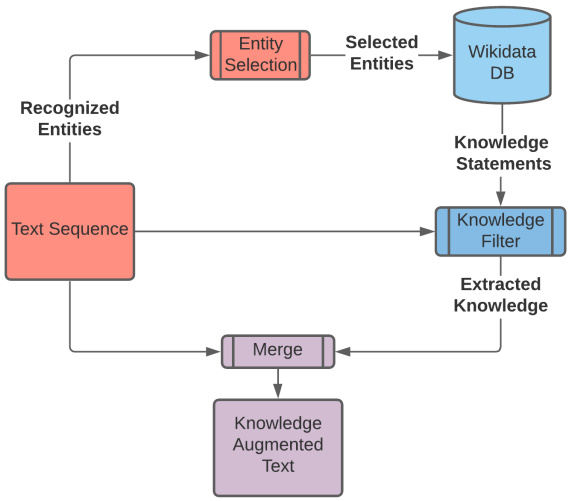



Figure 2.1: For each sequence in a language dataset, spaCy is used to identify the entities. Based on the entity selection criteria, a single entity is used to query the Wikidata database to acquire a set of knowledge statements. These statements are then filtered based on their relevance to the corresponding sequence. Lastly, the knowledge for each sequence is concatenated to the end of the original text sequence.

algorithm to a single entity. This entity is then used to query the knowledge base to return a set of knowledge statements. The knowledge statements are filtered and the remaining statements are concatenated to the end of the original text sequence. Once the entire dataset is augmented, it is ready to be used for training the knowledge transformer model.

**Dataset:** The WikiText-103 dataset [58] is used to train each of the models. After removing rows with no entities in the knowledge base, this dataset consists of a split with 628,965 (99.8%) training rows and 1,320 (0.2%) validation rows[2]. The WikiText language modeling dataset is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia. It makes sense to use this dataset for fine-tuning GPT-2 because WebText excludes Wikipedia articles from it's training dataset [88].

**Knowledge base:** Wikidata [120] is a free and open knowledge base of structured [10] Wikimedia data. While there is an API to access the official version hosted by Wikidata, this approach proves to be too slow to be used on large datasets. To reduce the time of each Wikidata query, a downloaded JSON data dump of the knowledge base can be converted into a database. While Wikidata supports many languages, only the English entities and properties are used.

In Wikidata, *items* refer to entities in the knowledge base, including people, topics, concepts, and objects. For example, the "1988 Summer Olympics", "love", "Elvis Presley", and "gorilla" are all *items* in Wikidata. A *statement* is defined as a relation between an *item* and a *value* by way of a *property*. *Statements* follow the resource description framework

[2]These are standard splits from Huggingface's datasets library at `https://huggingface.co/datasets/wikitext`

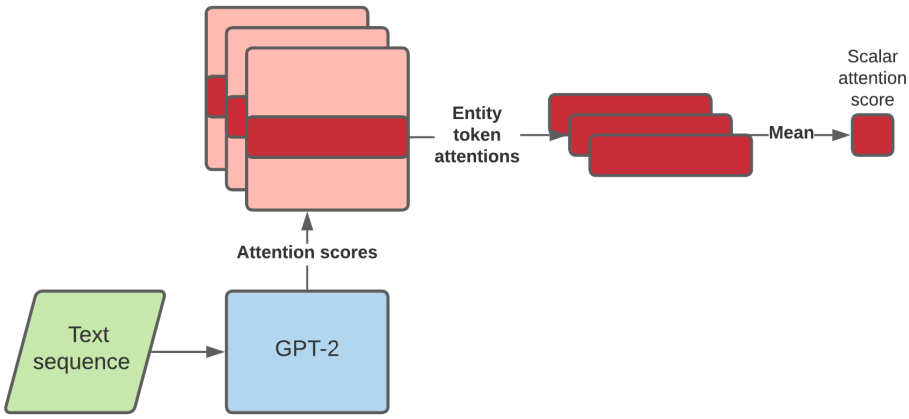



Figure 2.2: To compute an entity-specific score: a sequence is input to a pretrained GPT-2 model and the token attention scores are returned in an $n \times n \times h$ tensor, where $n$ is the sequence length and $h$ is the number of heads; the scores of each entity are isolated by selecting only the rows corresponding to the tokens of the entity in question; the scores are averaged across heads, the sequence length, and the entity tokens. The result is a single average attention score value for each entity in the sequence. This allows the entities to be sorted by their average attention score.

(subject-predicate-object) [62]. Generally, *values* are other *items* but can also be unknown or quantitative values.

**Attention-based Entity Selection:** The first step in the entity selection process is to identify words in the sequence that may have items in the knowledge base. This problem can be framed as a traditional named-entity recognition task to identify predefined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. For example, with the sentence "Apple is looking at buying U.K. startup for $1 billion", Apple, U.K., and $1 billion are considered entities. While there are many methods for named-entity recognition, spaCy[3] is used to extract the named-entities in each sequence.

Once a set of entities are identified in a text sequence, a single entity is included in the augmented dataset based on the entity selection criteria. In these experiments, an entity's average attention score is determined by inputting a sequence of text into a pretrained GPT-2 small model and returning the attention scores for each model layer. These attention scores are then averaged across layers, heads, sequence, and entity tokens—resulting in a single attention score for each entity in the sequence. These entities are then sorted by their (averaged) attention score to determine the maximum, median, and minimum attention-score entity. We interpret this ordering of entities to indicate their relative importance as determined by GPT-2 in the context of the input sequence. Figure 2.2 provides more details about how these attention scores are calculated.

**Knowledge Extraction:** In general, the knowledge extraction process consists of querying the knowledge base for information and then filtering that information based on

[3] `https://spacy.io/`

the input sequence. For these experiments, all of the knowledge is filtered out except for the description of the entity. This information is recorded in JSON format like so: *{label : description}*.

While this knowledge extraction process is simple, the purpose of this work is not to optimally select the best possible knowledge for a given input sequence. For now, it is sufficient to show that even naive semantic data can improve performance on language tasks. The task of developing more sophisticated knowledge extraction methods is left to future work.

**Merge:** The augmentation process is complete once the extracted knowledge is combined with the original dataset. This is done by concatenating the knowledge text to the end of the input text.

### 2.4 Experiments

The knowledge augmentation process is evaluated on a language modeling task. Since a language model is often indirectly asked to exploit the semantic knowledge it has acquired through pre-training, it stands to reason that language modeling would benefit from adding supplementary knowledge to the dataset.

#### 2.4.1 Knowledge-augmented Language Modeling

This section describes the details for handling a language modeling task with additional semantic knowledge added to the dataset. While modifications to a model's architecture are not always necessary, some changes may be required to allow the model to exploit this added knowledge.

**Knowledge Model**

GPT-2 is used as the base model for causal language modeling on each knowledge augmented dataset. The causal mask of Huggingface's [126] implementation of the standard GPT-2 architecture must be modified to allow the model to attend to the knowledge tokens at the end of the input (see Figure 2.3).

While the text tokens are masked normally, the knowledge tokens are always attended to. One could argue that this gives an unfair advantage to the knowledge model over the baseline GPT-2 model because the added tokens bias the model early on in the sequence. First of all, we argue that this bias is a good thing because it more closely resembles how a human might rely on their semantic memory while reading or listening. In spite of this, these results include experiments with a smaller, filtered dataset where additional contiguous text populates the knowledge tokens buffer (a sliding window over the entire dataset is not used). This filtered dataset only consists of rows that have excess knowledge tokens (333,753 train, 877 validation). Comparing this dataset with an identical dataset where knowledge tokens fill the knowledge buffer considers the benefit of added semantic knowledge vs. the benefit of additional textual context. We also include results using an attention mask which eliminates entity leakage from the added knowledge.

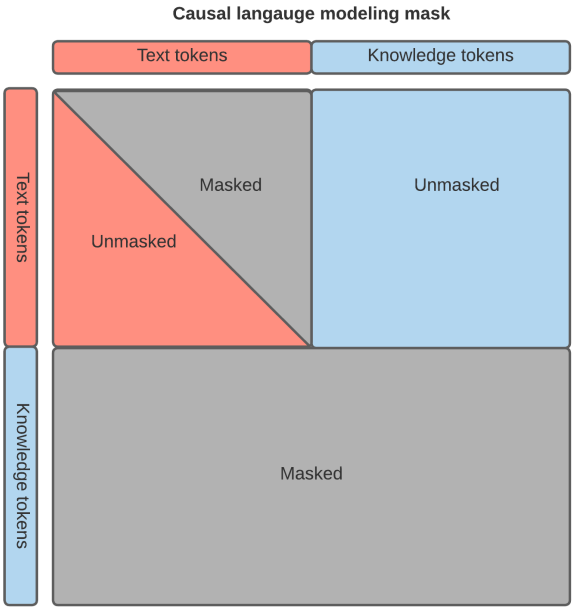



Figure 2.3: Modifications to the causal mask of Huggingface's GPT-2 to allow the text tokens to attend to knowledge tokens for each prediction. Recall that attention scores are calculated from each token to every token in the sequence, resulting in an $n \times n$ attention matrix. The text tokens are masked normally, with one less token being masked for each row until all but one of the tokens is unmasked (upper left quadrant). In the upper right quadrant, all of the knowledge tokens are unmasked to allow each unmasked text token to attend to the knowledge tokens. Since the model does not predict on the knowledge tokens, all tokens are masked on the lower half of the causal mask.

#### Baseline

The baseline for this experiment is an unmodified pretrained GPT-2 small model fine-tuned on the WikiText dataset. This is sufficient to observe the effect that additional knowledge has on language modeling. All of the models are trained using a batch size of 32, a learning rate of 1e−4, and the ADAM optimizer [41]. The early stopping criteria terminates training after three epochs of no improvement to the validation perplexity. The length of the text sequence is limited to 128 tokens while leaving a 64 token buffer available for knowledge tokens.

#### Entity Selection Criteria

These experiments focus on three primary variations of entity selection. As discussed previously, the average attention score for each entity is calculated using the attention scores output by a pretrained GPT-2 model (Figure 2.2). Given a list of entities ordered by attention score, the performance when using the maximum-, median-, or minimum-attention score as the entity selection criterion is compared. Based on which entity is selected, the corresponding description is retrieved from the knowledge base to form a knowledge statement. This knowledge statement populates the knowledge buffer as described previously. The four possible combinations of these primary variations (max/median, max/min, median/min, max/median/min) are also tested. The differences in performance of each of these variations provides insight into whether the entity selection criteria has an effect on the performance of a task.

### 2.5 Results

These results show that even simple knowledge augmentation can dramatically improve performance on language modeling tasks. At its best, the validation perplexity decreases by 81.04%. These improvements persist even as the number of model parameters increases and with a different model architecture (BERT) on a masked language modeling task.

#### 2.5.1 Knowledge Language Modeling

#### GPT-2 Small

Figures 2.4 and 2.5 illustrate how the best knowledge augmentation strategy (min attention) causes the baseline validation perplexity to decrease by 70.29%. Of the three knowledge augmentation strategies tested, min attention performed the best, followed closely by the median attention (0.19% worse), and then max attention (3.92% worse than min attention).

Based on these results, all of these knowledge augmentation strategies are effective in improving the performance of pretrained transformer language models on causal language modeling fine-tuning tasks. This increased performance is achieved without optimizing the knowledge augmentation process—instead, simply adding a *label:description* relation to the knowledge buffer. It stands to reason that the perplexity could be reduced further by adding more relevant data to the knowledge buffer.

While the difference in minimum validation perplexity between the min and median attention runs is small, the min attention runs consistently outperform the max attention runs.

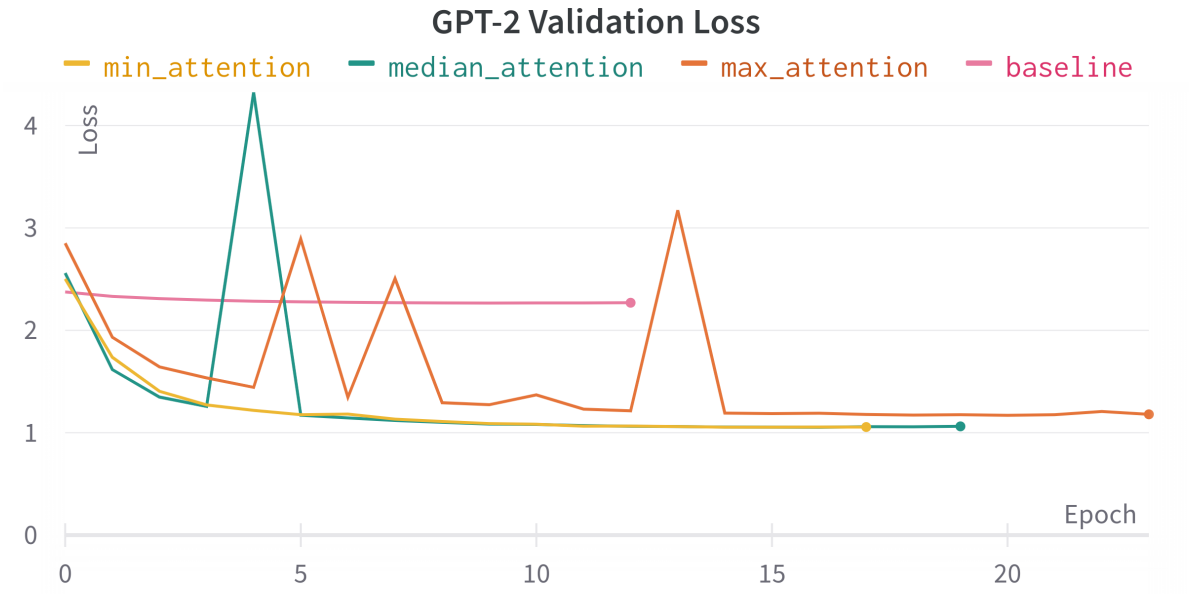



Figure 2.4: Validation loss curves for language modeling on GPT-2 small. The min attention knowledge augmentation strategy decreases perplexity by 70.29%.

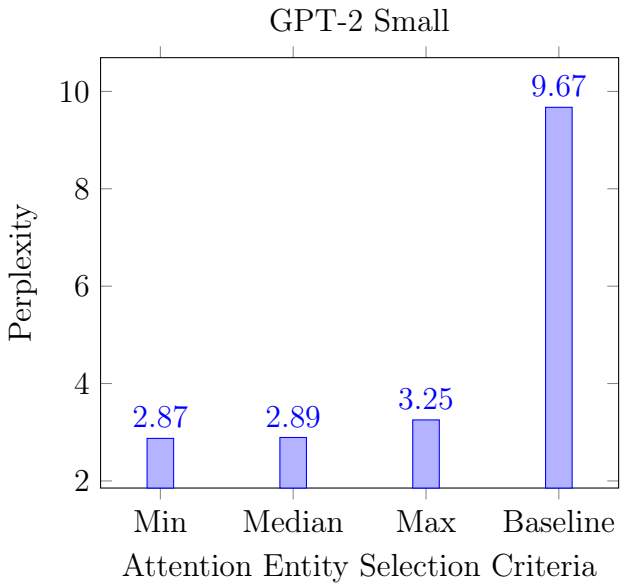



Figure 2.5: The validation perplexity for each augmentation strategy when language modeling on GPT-2 small.

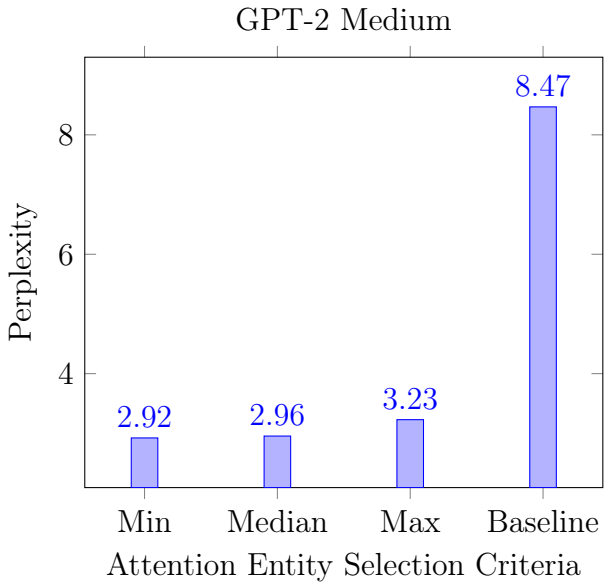



Figure 2.6: Validation perplexity for each knowledge augmentation strategy when language modeling on GPT-2 medium. Knowledge augmentation methods continue to improve performance even on larger language models.

This may be caused by GPT-2 assigning lower attention scores to entities it does not know very well and higher attention scores to entities it recognizes. By this logic, the additional data about an already common entity could be seen as redundant while the description of an unknown entity might be vital information.

### GPT-2 Medium

While GPT-2 small has 85M parameters, GPT-2 medium increases this by 313% to 354M. Despite these added parameters, the validation perplexity for GPT-2 medium only decreases by 12.46% when compared to GPT-2 small (compare baselines from Figures 2.5 and 2.6). However, adding min attention knowledge to GPT-2 medium decreases the validation perplexity by 69.78% from the GPT-2 small baseline. The difference between min, median, and max is comparable to the experiments with GPT-2 small.

The knowledge augmentation approach remains effective even as models get larger. This suggests that even very large models such as GPT-3 [8] could benefit from additional knowledge. In fact, the knowledge augmentation is more effective than increasing the model size since just adding min attention knowledge to GPT-2 small outperforms GPT-2 medium by 66.07%.

### Higher Order Combinations of Knowledge

Figure 2.7 shows the validation perplexity for each combination of knowledge augmentation strategies. As discussed previously, the best first order reduction, min attention, decreases the validation perplexity by by 70.29%. The best second order combination of min and max attention improves over min attention by another 7.37%. Finally, the combination of min,

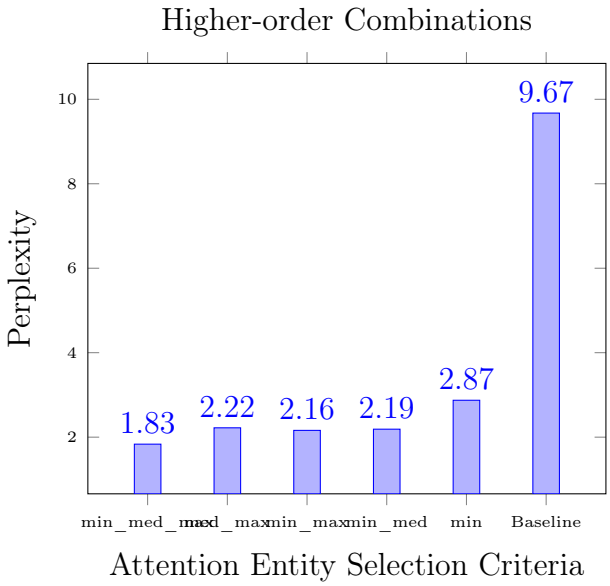



Figure 2.7: The validation perplexity for higher-order combinations of knowledge data when language modeling on GPT-2 small. Augmenting with minimum, median, and maximum knowledge entities yields the best results reducing the baseline validation perplexity by 81.04%.

median, and max attention knowledge improves over min_max attention by an additional 3.37% for a total of 81.04% improvement over the baseline. This demonstrates that a higher quantity of added semantic information consistently results in better generalization.

### Excess Text Tokens Versus Knowledge Tokens

Figure 2.8 compares the validation perplexity between filling the knowledge buffer with additional contextual text tokens or min_med_max knowledge tokens. For this dataset, knowledge tokens outperform the text tokens by 1.60%. This suggests that structured semantic data has a distinct advantage over additional textual context tokens.

### BERT

Figure 2.9 once again shows the minimum validation perplexity for the primary knowledge augmentation strategies. In this experiment, only the median and max attention knowledge augmentation runs outperform the baseline, with the max attention variation resulting in the greatest percentage decrease of the baseline perplexity (11.23%). While this reduction in perplexity is not as large as with GPT-2, there are several contributing factors which may explain this. First of all, BERT is pretrained on the English Wikipedia corpus [25] while GPT-2 excludes it. This means the pretrained BERT is already relatively close to it's training limit when fine tuning begins. Secondly, because BERT is a bidirectional model, it can attend to all unmasked tokens in the sequence for each prediction. This makes the added knowledge less useful in disambiguating the subject of the sentence when compared with causal language modeling—especially early on in the sequence. Also, since BERT only

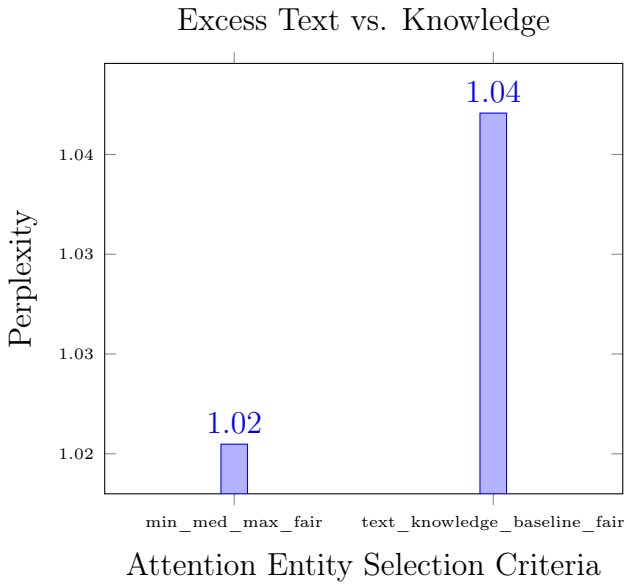



Figure 2.8: Validation perplexity when language modeling on GPT-2 small with comparison between filling the knowledge buffer with excess text tokens and filling it with min_med_max knowledge tokens for a dataset where each row has excess tokens.

predicts on 15% of tokens, these perplexity values are not directly comparable. Taking all this into account, these results continue to validate the efficacy of these methods even across model architectures.

### Entity Leakage

As shown in Figure 2.2, the knowledge buffer is always visible for each step in the decoding process. However, since the knowledge tokens contain at least one entity label that occurs in the sequence, this inherently leads to information about upcoming tokens being leaked to the model. Therefore, the knowledge model has an unfair advantage compared to the baseline model in that it is easier to predict at least one entity in the sequence. In order to evaluate the extent to which the knowledge model benefits from entity leakage, additional experiments use an alternative masking approach which seals any potential leaked information. This dynamic knowledge mask causes the knowledge tokens to remain hidden until the last token of the entity label is exposed in the input sequence. All other parts of the dataset augmentation and training processes remain the same.

Results in Figure 2.10 show each of the entity selection criteria still reduce validation perplexity from the baseline even without entity leakage. Despite the expected drop off in performance from removing entity leakage, max attention beats the baseline by 58.69%. It should also be expected that min and median attention suffer slightly more than max attention due to the fact that the min and median entities are more likely to be positioned toward the end of a sequence. Overall, these results verify the efficacy of these knowledge-augmentation strategies independent of any entity leakage.

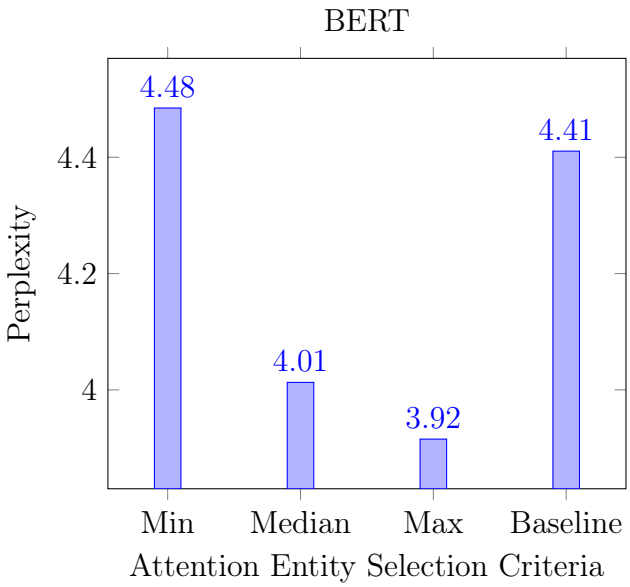



Figure 2.9: Validation perplexity when language modeling on BERT. Knowledge augmentation continues to improve results on BERT despite its being pretrained on Wikipedia.

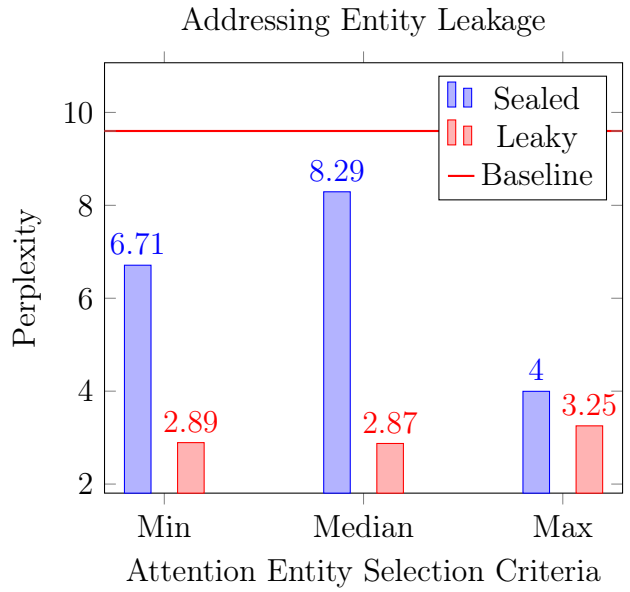



Figure 2.10: Validation perplexity when language modeling on GPT-2 small with and without entity leaking.

### 2.6 Discussion

The purpose of these experiments was to determine whether knowledge-augmented datasets are effective in improving a model's semantic memory. The significant improvement demonstrated on both language modeling shows that knowledge augmentation does make a difference. One possible reason for this could be that statistical language models rely so heavily on the context around the word that the definition of the word itself remains a little too obscure. This may get to the point where words used in an unfamiliar context, possess little meaning and confuse the model. This would explain why including additional semantic information provides a useful bias that keeps unfamiliar words in context. This is further demonstrated by the fact that the min attention entity generally outperformed max and median attention entities. Assuming that the min attention entity is deemed least important by GPT-2, the added knowledge for this entity appears to give new relevance to overlooked or unfamiliar data.

Another way to think about knowledge augmentation is as a form of prompt engineering – where a prompt is automatically generated to help the model with the task at hand.

In these experiments, the entity identification, selection, and knowledge extraction process is relatively simple. In future work, this entire process would be self-directed where a single model learns to use the knowledge base to best suit the task at hand.

As language models becomes more important to society, challenges will arise where language modeling will fall short if it is solely reliant on statistics. Ethical concerns regarding transformer models spreading misinformation and bias have already harmed the reputation of popular language models such as GPT-3 [2]. The principles discussed in this work will directly enable ways of biasing language models away from the dubious ideas present in "wild" training data and towards a shared reality present in curated knowledge bases. Code for this work is available at `https://github.com/rmorain/symbolic_semantic_mem`.

# Chapter 3

## Are Language Models Unsupervised Multi-domain CC Systems?

I hereby confirm that the use of this article is compliant with all publishing agreements.

Robert Morain, Branden Kinghorn and Dan Ventura, "Are Language Models Unsupervised Multi-domain CC Systems?" Proceedings of the International Conference on Computational Creativity, 2023, pp. 39-43.

**Abstract**

Recently, ChatGPT has grown in popularity due to its ability to generate high quality text in a wide variety of contexts. In order to determine whether ChatGPT threatens to undermine the need for traditional CC systems, ChatGPT's ability to generate textual creative artifacts needs to be formally analysed. To do this, we constructed a survey that compares artifacts generated by traditional CC systems with corresponding artifacts generated by ChatGPT. Both types of artifacts are also evaluated independently on how well they possess certain desirable characteristics. Overall, the survey shows that artifacts generated by ChatGPT are preferred 36.84% ($p = 0.014$) more often and rated higher by 0.5 mean Likert scale points ($p = 0.0004$). These results indicate a need to reconsider the purpose and approach of traditional CC systems going forward.

### 3.1 Introduction

Computational creativity (CC) researchers often create applications that address creativity in specific domains such as stories [77], poetry [5], or puns [91]. These CC systems often introduce novel methods for generating creative artifacts such as templates, rules, or machine learning models. The authors then evaluate these generated artifacts either automatically or by way of a user survey. Recently, ChatGPT [66] has demonstrated impressive text generation abilities. In this paper, we aim to evaluate ChatGPT's ability to generate creative artifacts by comparing ChatGPT's artifacts to artifacts generated by domain specific CC systems. While the scope of these experiments could include other modalities such as images [89], this paper focuses on textual creative artifacts.

This paper uses a definition of creativity that focuses on the generated artifact rather than on the process by which is created [125].

### 3.1.1 Motivation

As statistical large language models improve, the need for domain-specific CC systems requires further consideration. If traditional CC systems are to remain relevant, they must offer distinct advantages over models like ChatGPT and its successors. ChatGPT implicitly learns many language related tasks through the general tasks of autoregressive language modeling [86] and fine-tuning with human feedback [71]. The extent to which these abilities overlap with traditional CC systems is the central focus of this paper. Another topic related to this work is the philosophical question surrounding the creative limitations of language models.

All of the data used for this paper including prompts, survey results, and analysis is available online.[1]

## 3.2 Methods

The survey used in this paper includes artifacts from a selection of CC systems covering four textual domains: puns, six word stories, poetry, and short stories. These systems are JAPE [91], PAUL BOT [114], Movie Title Puns [34], MICROS [106], HIEROS [102], a GPT-2 model fine-tuned on six-word stories [14], mBART for poetry generation [5], TwitSong 3.0 [46], PoeTryMe [65], MEXICA [77], INES [19], and Chronicle [83].

Notice that this survey does not discriminate against CC systems that use language models. The focus of the survey is not to determine whether language models are useful in CC systems, but rather to evaluate ChatGPT's creative text generation abilities compared to existing CC systems regardless of the methods used in the system.

While many potential systems could have been selected for the survey, these systems were chosen based on a combination of factors: the availability of artifacts present in the paper; the popularity of the system in the CC community; suitability to the domain; and recency of the paper's publication. Interested readers should refer to these papers for more details about their methods and implementation.

A set of twelve artifacts was compiled by selecting one artifact from each paper, to reduce the possibility of a single system under performing and thus skewing the results. Because the highest scoring artifact is not reported in the papers, the artifact expected to perform best according to the authors' subjective judgement was chosen.

Given the set of selected artifacts from CC systems, corresponding artifacts with similar subject matter were generated using ChatGPT. [2] ChatGPT was prompted to create an artifact from a particular domain (pun, six-word story, etc.) that included the same subject matter as the original artifact. Table 3.1 provides an example for how these artifacts were generated. This process facilitates the comparison of artifacts based on quality rather than other factors such as preference of subject. In some cases, when the generated artifact was too long or did not possess the correct subject matter, ChatGPT was iteratively prompted to generate a suitable artifact. Otherwise, the first artifact generated was selected. Artifacts were also screened for plagiarism by searching the web for exact copies.

[1] `https://tinyurl.com/s4k5pwry`

[2] At the time of this experiment in April 2023, ChatGPT uses GPT 3.5 (See release notes: `https://help.openai.com/en/articles/6825453-chatgpt-release-notes`).

| | |
|---|---|
| CC system artifact: | How is a nice girl like a sugary bird? They are both sweet chicks. |
| Prompt: | Think like a punster. How is a nice girl like a sugary bird? |
| ChatGPT: | Here's a possible punning answer to your riddle:<br>How is a nice girl like a sugary bird?<br>They're both tweethearts! |

Table 3.1: To create an artifact using ChatGPT with the same subject as a CC system artifact, a prompt with the appropriate domain and subject matter is provided to the model. The generated artifact is manually extracted. ChatGPT does not receive the original artifact in the prompt.

Next, a survey was created to evaluate the artifacts based both on reviewers' preferences and characteristics used by various authors to evaluate the corresponding CC systems. To evaluate preferences, reviewers are asked to choose between a CC system artifact and the corresponding ChatGPT artifact, in a side by side comparison. Reviewers also had the option to mark "no preference". The reviewers were not made aware of which artifact came from a specialized CC system and which came from ChatGPT. To evaluate artifacts based on their characteristics, reviewers rated each artifact based on how well they possessed each characteristic on a Likert scale (1: strongly disagree, 2: somewhat disagree, 3: neither agree nor disagree, 4: somewhat agree, 5: strongly agree). For puns, the evaluation characteristics are "funny," and "surprising"; for six-word stories, "coherent" and "impactful"; for poems, "meaningful" and "emotional"; and for short stories, "entertaining" and "surprising". These characteristics were selected from the evaluation criteria used by the original authors to evaluate the CC systems. In addition, artifacts from all four domains are also rated on how creative they are perceived to be.

The survey was distributed online through Facebook, Instagram, Twitter, and Reddit. On Reddit, the survey was sent to the r/ArtificialIntelligence, r/MachineLearning, r/deeplearning, and r/ChatGPT subreddits. The survey does not ask for respondents to identify themselves or to rate their own knowledge of AI or CC; therefore it is unknown whether the reviewers are experts or not. The survey is randomized such that the questions and answers appear in random order.

### 3.3 Results

Responses from 148 individuals resulted in an average of 39.5 responses to each question in the survey. Figure 3.1 shows reviewers' overall preferences across all domains and systems. The artifact produced by ChatGPT is preferred over the related CC system artifact 63% ($p = 0.014$)[3] of the time. However, the difference in terms of the characteristic evaluation of the two types of artifacts is relatively small. Figure 3.2 shows a difference of 0.50 Likert scale points ($p = 0.0004$), favoring the ChatGPT artifacts. Using the common significance threshold of 0.05, both of these results are statistically significant.

[3]Significance is calculated using a paired sample t-test.

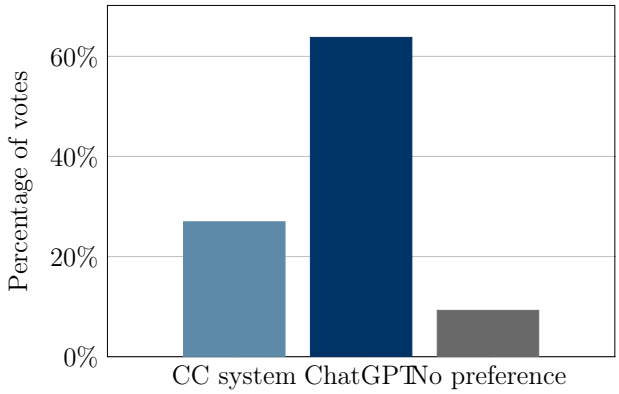



Figure 3.1: Reviewers' preferences in a one-to-one comparison between CC system generated artifacts and corresponding ChatGPT generated artifacts. These votes are aggregated across all domains and systems.

Figure 3.3 shows reviewers' preferences broken down by the four domains and aggregated across the three systems in each. For each domain, ChatGPT gains at least 61% of the votes. ChatGPT received the lowest percentage of votes in the poetry domain and the highest in the short story category with 77% of the votes. However, Figure 3.4 shows that the characteristic scores for the ChatGPT artifacts are relatively close to those for the original CC system artifacts. ChatGPT's lowest mean Likert scale score is in the pun domain with a score of 2.93 which is 0.10 points lower than the CC systems' score. The domain with the largest difference is the six-word story category with a margin of 0.62 points in favor of the ChatGPT artifacts.

The preferences for each artifact generated by their respective CC system along with the ChatGPT generated counterpart is shown in Figure 3.5. For each system, the ChatGPT artifacts are preferred, with the exception of artifacts produced by PAUL BOT and Chien 2020. It is interesting to note that the INES system did not receive a single vote.

Figure 3.6 shows the mean Likert scale score for each artifact. The highest score overall belongs to [14] which was generated by GPT-2 fine-tuned on a dataset of six-word stories. The characteristic evaluation scores usually correlate with the reviewers' preferences in that preferred artifacts have a higher score, with the exception of Chronicle which is preferred less but has a higher characteristic evaluation score than its ChatGPT counterpart.

For the characteristic evaluations, we can measure agreement between reviewers as a way to further assess our ability to be confident in the survey results, and this inter-rater agreement can be measured using Krippendorff's alpha [44]. Across all systems and domains (cf. Fig 3.2), reviewer agreement produces $\alpha = 0.291$. Figures 3.7 and 3.8 show reviewer agreement broken down by domain (cf. Fig 3.4) and system (cf. Fig 3.6). Each of these values fall well below the recommended threshold of $\alpha \geq 0.8$ that would suggest reliable inter-rater agreement on preference for one system over another.

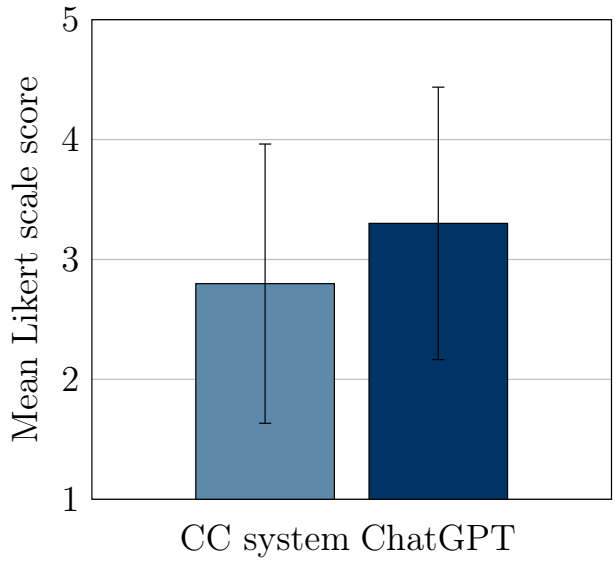



Figure 3.2: Characteristic evaluation of generated artifacts aggregated across all domains and systems.

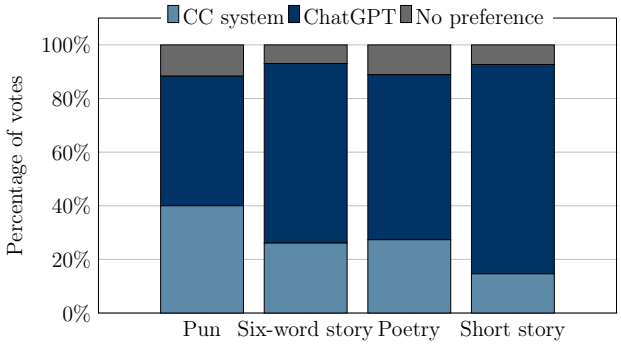



Figure 3.3: Reviewers' preferences aggregated across systems but broken down by domain.

### 3.4 Discussion

The results seem to indicate that ChatGPT is able to generate artifacts that are just as good or better than the CC systems. This is similar to results found in [86] which shows that training a model on a general task like autoregressive language modeling leads to improved zero-shot performance on several downstream tasks as well. In this case, the data show that ChatGPT generalizes to creative tasks by outperforming CC systems overall, as well as at the domain and individual system level. The statistical significance of these results suggests that ChatGPT artifacts are likely to be preferred to and rated higher than (current/traditional) CC system artifacts.

While the results show that ChatGPT is capable of matching or surpassing CC systems in terms of the characteristic evaluation across all domains (Figure 3.4), the relative difference between CC system and ChatGPT artifacts is not as large as in the direct preferences analysis (Figure 3.3). In addition, the inter-rater agreement at the overall, domain, and system level is

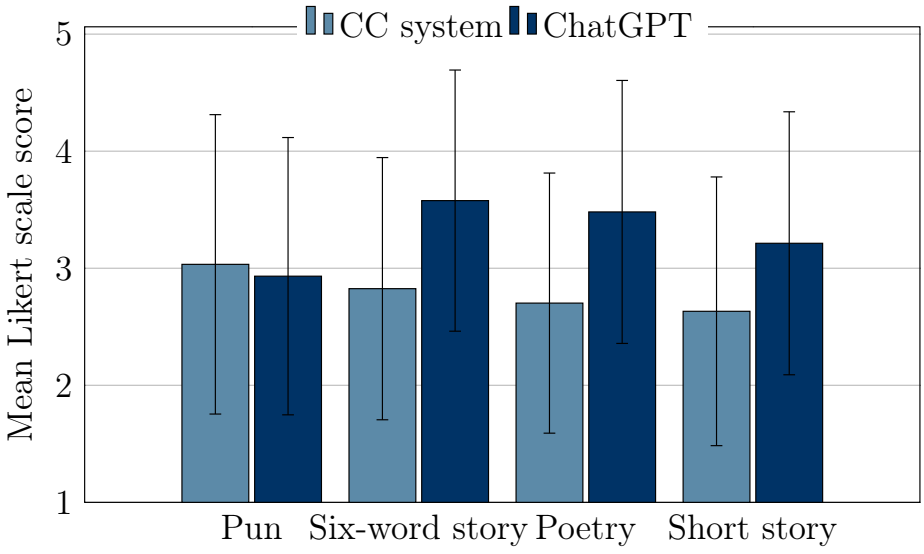



Figure 3.4: Reviewers' characteristic evaluation of artifacts in each domain, aggregated across systems.

well below the recommended threshold for reviewer agreement, suggesting that characteristic evaluation does not completely explain reviewers' preferences.

One reasonable explanation for this is that an artifact only has to be slightly better in order to be preferred. Although, the presence of a "no preference" option provides confidence that there is a real difference in preference between the artifacts, even if that preference is small.

It is also possible that the criteria used in the characteristic evaluation fail to capture all of the reasons why reviewers prefer an artifact. For example, large language models like ChatGPT are very capable of generating fluent text even if the content of the text is nonsense. In addition, there may be other positive characteristics that ChatGPT includes in its artifacts, such as accessibility to a general audience or even other domain specific characteristics.

It is also reasonable to conclude that the characteristic evaluation is reliable—reviewers generally prefer the ChatGPT artifacts, and while the difference between the artifacts in terms of their character evaluation is not large, the significance testing provides confidence that this difference is, in fact, real. Also, it is important to remember that the artifacts selected for the survey that came from CC systems are (presumably) the best those systems have to offer. On the other hand, ChatGPT's artifacts are not cherry picked and most of the artifacts were generated with a single non-engineered prompt. Therefore, it may be argued that these results may represent a comparison of the floor of ChatGPT's abilities to the ceiling of (traditional) CC systems' abilities.

### 3.4.1 Implications and Future Work

The findings of this survey do not discount the work of CC researchers. Rather, their accomplishments with significantly fewer resources indicate that many of these traditional

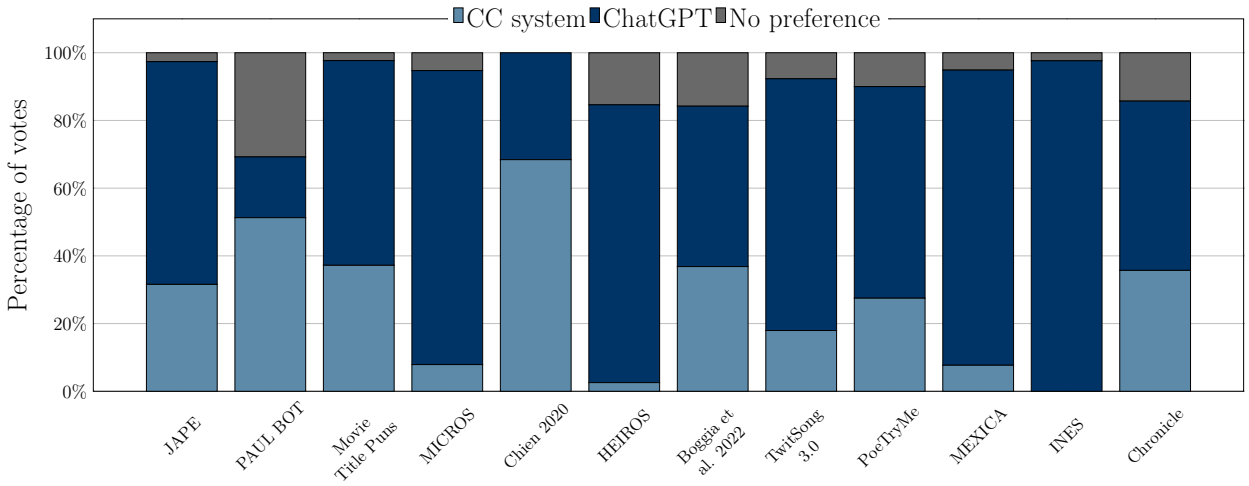



Figure 3.5: Reviewers' preferences broken down by the system that generated each artifact.

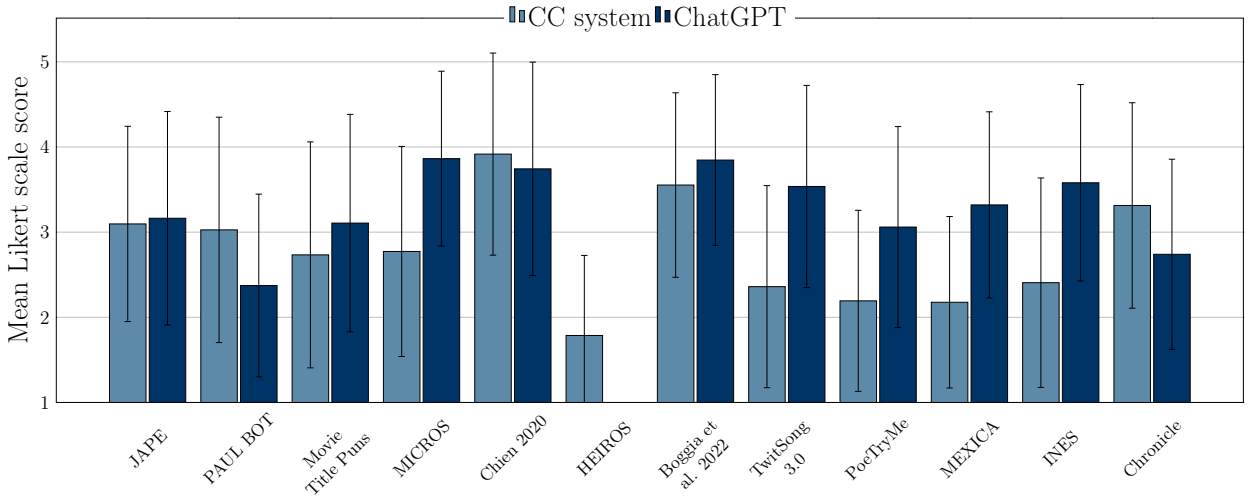



Figure 3.6: Reviewers' characteristic evaluation of each artifact.

CC systems are truly ahead of their time. It is also possible that the methods demonstrated by these systems applied at the scale of ChatGPT may outperform ChatGPT.

The purpose of this paper is to spark debate about the creative limitations of language models like ChatGPT and CC systems in general. Given that this level of performance comes from a general language model like ChatGPT means that the purpose and approach of domain-specific CC systems needs to be carefully considered. At the very least, ChatGPT should be used as a baseline when evaluating CC systems going forward.

ChatGPT represents a paradigm shift in terms of interactivity in creative systems. In these experiments, interactive prompts serve to constrain the system to produce corresponding artifacts that are comparable to their CC system counterparts. ChatGPT's ability to do this successfully demonstrates the system's robustness and ease of use. It also suggests a possible move away from fully autonomous systems towards more co-creative solutions (though this certainly doesn't preclude fully autonomous systems in any way, of course.)

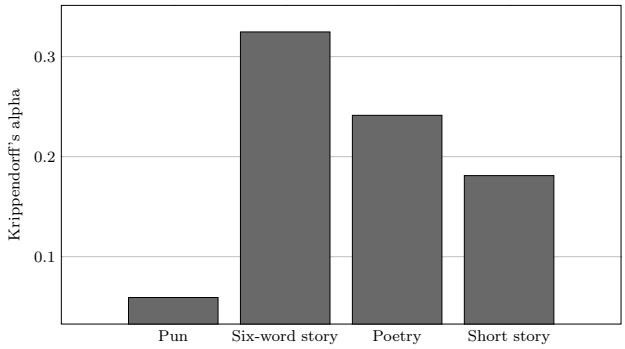



Figure 3.7: Agreement between reviewers by domain.

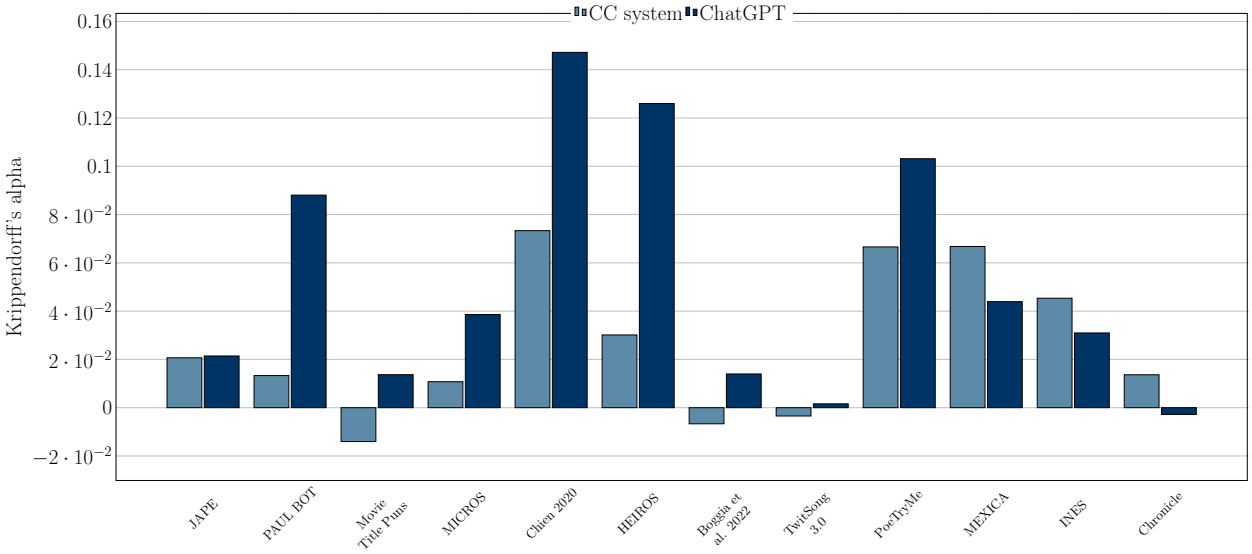



Figure 3.8: Agreement between reviewers by system.

These results also highlight an opportunity to improve the performance of language models on creative tasks. While the ChatGPT artifacts are preferred, the overall characteristic evaluation shows that reviewers still have a generally neutral attitude toward the artifacts. It is not yet clear from where these improvements will come, but it is possible that some help may be found in traditional CC approaches.

# Chapter 4

## A Development and Teaching Framework for Codenames

I hereby confirm that the use of this article is compliant with all publishing agreements.

Robert Morain, Brad Spendlove and Dan Ventura, "A Development and Teaching Framework for Codenames" Proceedings of the International Conference on Computational Creativity, 2025.

### 4.1 Introduction

Methods for the external evaluation of creative computational (CC) systems have long been the subject of debate, and thinking on the subject continues to evolve [7, 12, 37, 47, 74, 75, 92, 117]. In a significant sense, this question of how to evaluate CC systems is foundational in that it plays a large role in specifying the goals of the field and what it means to accomplish them. Historically, many evaluation methods have aimed to measure inherently subjective creative attributes, such as quality, novelty, typicality, and surprise, whose interpretations and relative importance vary across individuals and domains. Given that accurate evaluation is critical for advancement and maturation of the field, the limitations imposed by the approximation of subjective features represent a major challenge. To address this issue, Spendlove and Ventura proposed that competitive language games could serve as creative tasks with well-defined goals [104]. The win/loss outcome of the game eliminates the need to approximate subjective measures by providing an objective proxy measure of creativity. The well-defined goals of the game do not reduce the need for creativity on the part of the system. Rather, the constraints of the task provide an opportunity to highlight creative gameplay.

In particular, the authors explore the design and evaluation of CC systems with the game Codenames [15], a word-based guessing game. In the game, two teams of players are tasked with identifying which of 25 cards, drawn at random from a large deck, belongs to their team. One member of each team serves as the spymaster. They are given a secret key that shows which words belong to each team. The teams then take turns in which the spymaster gives a one-word clue, accompanied by a number, that attempts to communicate a subset of the team's words to their teammates. The clue word must relate to the meanings of the word cards, and the clue number represents how many cards are intended to be related to the clue. Their teammates then guess one card at a time. If they correctly guess a word that belongs to their team, they may continue guessing, otherwise the turn is over. Wrong guesses may reveal opponents' words or even instantly lose the game. The challenge for the spymaster lies in the puzzle of coming up with a clue that strongly relates to their team's cards without unintentionally relating to any others.

Creative domains are characterized by their extremely large combinatorial spaces. Playing the spymaster role constitutes a creative task given the vast number of possible solutions (semantic graphlets that indicate positive and negative relationships between candidate clue words and the target words to be guessed) and the complexity of language inherent in the gameplay. Designing an agent to play the game is challenging, as it requires both a deep understanding of language and potentially some kind of theory-of-mind-like model of the way teammates understand and think about language. The agent must employ creativity to efficiently select clues from a large solution space for each possible game state in a (typically) limited amount of time. These requirements position Codenames as a challenging creative task that can serve as an ideal mechanism for the evaluation of creativity through gameplay [105].

To kickstart the research community's interest in games as creative tasks, an introductory workshop was held as part of *ICCC24*. In this workshop, participants were introduced to the necessary theory and mechanics of gameplay before being invited to modify a provided agent or create a new one. A web application with a Python client developed for the workshop helped participants build and evaluate their agents. This enabled participants to examine the behavior of an agent, come up with new ideas for how to improve it, and efficiently build a working prototype of their solution. The workshop outcomes provided further evidence of the usefulness of Codenames as a platform for CC research.

During the workshop, participants demonstrated the ability to successfully and rapidly design, implement, and iteratively improve a creative agent to play Codenames using the resources provided. This inspired the idea to evaluate the resource's effectiveness in CC education. To do this, a study was developed for a graduate-level introductory CC class. In the study, students were assigned to develop a creative agent using an improved version of the Codenames resource developed for the workshop. Students then engaged in an iterative development process, during which they had the opportunity to evaluate their agents in a tournament, identify areas for improvement, and redesign their agents. Finally, students participated in a second tournament to evaluate their improved agents. Throughout the experience, students completed two surveys designed to gain insight into the effectiveness of the resource for the development process and to gauge student's understanding of fundamental CC concepts.

Motivated by the positive experiences of the workshop participants and graduate student study participants, this paper addresses the need for expanded access to game-based creative evaluation metrics by publishing this open-source resource for building and evaluating agents that play Codenames. We anticipate that this resource will serve the research community in many ways, and we put forward these two use cases as examples of its utility. We have found that the game's engaging nature draws people in, sparks ideas, and motivates them to design agents to play it. This framework leverages those unique qualities to make the development process both enjoyable and effective in research and pedagogical settings.

This paper contains a report on the workshop, a description of the Codenames framework and web application, the results of the study, and discussion of future use cases and applications. By providing a practical tool for game-based creativity research, this work aims to expand access to well-defined evaluation metrics and encourage further exploration

of competitive language games as a testbed for CC systems. All of the code used for this paper is available on GitHub.[1]

### 4.2 Workshop Report

The purposes of the workshop were to introduce our work to the broader research community, engage in discussion of the creativity of gameplay and game-playing agents, and host a Codenames agent code jam. To support these activities, we developed a programming and UI framework for playing Codenames with human and AI players. The framework includes a default spymaster agent capable of generating clues for any Codenames game state without additional setup. The game UI is a user-friendly web interface that makes it easy to create and play Codenames games. These tools facilitated explanation, discussion, and experimentation at all levels of the workshop.

At the beginning of the workshop, we played a sample game of Codenames with the participants via the web application. The web application automates game rules such as turn passing and scoring, which helped the participants quickly understand the spymaster and guesser tasks. After playing a single game of Codenames, the participants already had ideas for how to develop AI agents to play the game. We discussed how they, as human players, approached clue-giving and guessing, and how those strategies could inform the design of an automated agent.

A large portion of the workshop was dedicated to a code jam, in which participants developed their own Codenames agents. Participants leveraged our Codenames framework and UI to quickly prototype working agents that could play against humans and one another. This allowed them to actually implement and test the ideas they shared in the discussion and see how they performed in the game. Their agents' designs included strategic selection of which subset of cards to relate to a clue, querying large language models to generate clues, and clustering the different colored cards using $k$-means. Participants reported that this exercise was an enjoyable and interesting way to interact with a creative task that was new to them and pointed to the framework as enabling this positive experience. The framework allowed them to focus on the core task of developing an agent that can create clues, without spending unnecessary time writing boilerplate or interface code.

In our concluding discussion, participants shared their takeaways from the workshop. We discussed whether playing Codenames is a creative task, the potential benefits of studying games as creative tasks, and how Codenames' focus on language could cross over to other language-based creative domains. We were pleased by the positive reaction to the workshop and encouraged that our Codenames framework facilitated rapid prototyping and experimentation.

### 4.3 Resource Description

The code base for the Codenames development framework includes a game server, a web client with a graphical user interface (UI), and Python clients for the spymaster and guesser

[1] `https://github.com/rmorain/codenames-ai-client`
`https://github.com/rmorain/codenames-workshop`

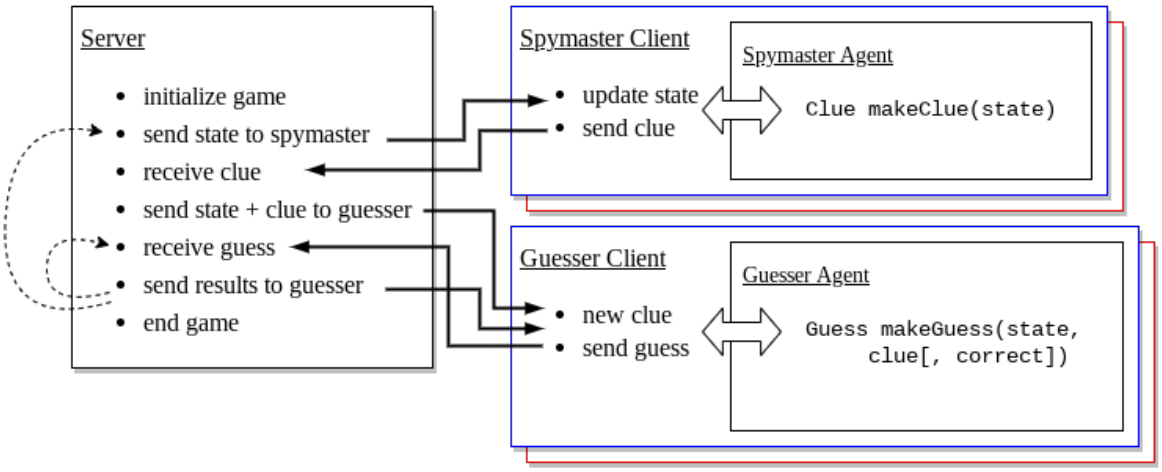



Figure 4.1: The server/client/agent architecture of the workshop codebase, including information flow between the components. Note that there are two spymasters and guessers, one pair each for the red and blue teams.

roles. Both the web UI and the Python clients connect to the same game server, allowing for gameplay with any combination of human players and AI agents in the various game roles. The Python clients each wrap an agent program that makes the gameplay decisions, marking a clear division between the creative agent code and the server interface code.

The primary use cases for this code base are playing Codenames as a human player and modifying the agent code to implement new creative play agents. Neither of these requires knowledge of or modification to the client/server code. We host a public instance of this server here: `https://mind.cs.byu.edu/codenames`, where anyone can connect and play via browser or client.

The backend development framework is divided into five components: the server that runs games, a spymaster agent, a spymaster client, a guesser agent, and a guesser client. The two types of clients facilitate communication between the agent and the server. See Figure 1 for a diagram of the complete architecture, including information flow between components. These components are described in detail in this section, and some gameplay details that were left out of the description of Codenames above are included here for completeness.

The server tracks the state of the game in progress, communicates that state to the clients, listens for their moves, validates those moves, and updates the internal game state. This state can be retrieved by either the web UI or the client code. This section focuses on the latter. Codenames is not a complex game to represent computationally. At the beginning of a new game, the board of 25 word cards is drawn from the deck of 400 (the list of which is available online). A coin is flipped to decide which team goes first, red or blue, then the first team is randomly assigned 9 cards and the other 8. The remaining cards are neutral, except for one assassin card chosen at random. This completes the starting board state, and the game is ready to play.

Each new game is assigned a unique four-letter code that clients use to join that specific game session on the server. Four clients must connect to the server: the two spymasters and the two guessers. When playing the game, each team may have any number of guessers, but

from an information perspective they act as one unit when choosing words to guess, so one client suffices.

The server sends the game state to all clients, including the secret key to the spymasters, and waits for the current team's spymaster to return a clue consisting of one word and a number. The server then sends the clue to the guesser client. The guesser can guess one word card at a time, after which the server updates the game state by revealing the hidden identity of that word card (red, blue, neutral, or assassin). If the guess is correct (revealed to belong to their team), the guesser may guess again, up to a maximum of one plus the clue number. If the guess is revealed to be the assassin, the game immediately ends, and the guessing team loses the game. Otherwise, if the guess is incorrect or the team is out of guesses, play passes to the other team and the process repeats.

Thus, the server requires the following functions:

- Initialize game, update state
- Transmit game state to current spymaster
- Receive clue from spymaster, update state
- Transmit game state and clue to guesser
- Receive guess from guesser, update state
- Transmit result of guess to guesser
- End game, update state

The clients handle communication between an agent and the server, so the clients' required functions mirror the server's. The spymaster client receives the game state from the server and passes it to the spymaster agent, which returns a clue word and number. The client transmits that to the server and waits until it hears from the server again to get the next clue.

The guesser client has two similar information flows. In the first, it receives the clue and current game state from the server, passes them to the guesser agent, gets a guess back, and transmits it to the server. In the second, it receives the result of the previous guess from the server and passes it to the agent, sending another guess back if necessary.

The spymaster client requires the following functions:

- Receive updated game state
- Submit clue

And the guesser client requires these functions:

- Receive game state and new clue
- Submit guess

The agents' functions follow from the description of their clients. The spymaster agent has a `makeClue` method that accepts a game state and returns a clue. The guesser agent has a `makeGuess` method that takes a game state and clue and returns a guess. The basic

game state data consists of the list of word cards and a list of the revealed identity of each word card (if any). Additionally, the spymaster receives the secret key showing all the cards' colors and the guesser receives the current clue and number of guesses remaining. The clients are very thin wrappers around the agents but are engineered this way to allow for simple modification to or replacement of the underlying agent. This separates the client/server communication logic from the agent so that its developers can focus only on the relevant creativity task.

To run the provided automated clients, the user only needs to provide the game code and team color. The client connects to the game server, then repeatedly polls the current game state, calling upon its agent module to generate a game action when required.

Finally, the code base also includes a web UI client for human gameplay that connects to the server using the same interface as the automated clients. This client presents a UI that mimics the layout of the physical game and through which users can input the required game actions. The game state is represented by a grid of word cards that are either uncovered, showing the word, or covered with the appropriate color after they're guessed. One client is used for both the spymaster and guesser role by either prompting the spymaster for a clue word and number or allowing the guesser to click on a word card to guess it when prompted. This simple interface is low overhead and allows human players to play with the AI agents, bringing more richness and interactivity to the application. The web client also provides an observer mode which simply displays the game state without allowing for any game action input. This is useful, for example, for displaying the game to spectators while others play the game.

The entire code base is public, allowing future users to reuse or modify any portion necessary. The server can be run locally, or games can be run on our public server. At the same time, the code base is designed to require only minimal modification to run a custom gameplay agent. The programmer must only replace the appropriate agent module within the client. They will then be able to play their agent against humans or other agents without modifying the client, server, or web UI code.

### 4.3.1 Example agents

To complete the code base, we include two basic gameplay agents, a spymaster and a guesser. These agents allow the clients to be run out-of-the-box without writing any code, and they serve as a reference implementation for the simple client-agent interface. Furthermore, the agents were designed to be simple to understand even for relatively inexperienced programmers, providing a jumping-off point for their own modification and improvement.

Both example agents are powered by word embeddings, which are vector representations of words that capture their semantic meaning [60]. These high-dimensional vectors enable computers to understand and process natural language to facilitate many tasks such as sentiment analysis, machine translation, and text classification. There exist straightforward analogues between vector operations and the cognitive task of comparing the relationships between words, which makes them a natural fit for Codenames agent development.

The cosine similarity between two word embedding vectors corresponds to the semantic similarity of the words those vectors represent [70]. By comparing the similarity of a clue word candidate to the word cards, the spymaster agent determines whether that clue will

effectively relate to those cards. This information is used to inform the choice of a clue that is similar to the team's words and dissimilar from the others. Conversely, the guesser agent calculates the similarity between a given clue and the word cards and guesses the one(s) that most closely relate.

These simple agents are complete agents in that they can play the game, and they use a reasonable natural language understanding approach to do so. They perform reasonably well in-game, but may often generate unclear clues. Of course, this is by no means the only way to design these agents, which is our motivation for publishing this code base. These starter agents will serve the needs of users with widely differing programming skills and AI or NLP backgrounds.

### 4.4 CC Pedagogy Study

Computational creativity in education is beginning to gain traction, both as an approach to teaching creativity and computational thinking [26, 39, 131] and as a first-class pedagogical subject itself. In particular, Ackerman et al. discuss the importance of CC education in the broader context of AI and provide guidance for instructors on how to approach CC pedagogy. Further, they suggest the benefit of students engaging in introductory assignments to reinforce the material and develop their own ability to develop creative systems [1]. Ventura provides an introductory guide for how students (as well as researchers) might approach building such CC systems by identifying their essential components [118]. This Codenames framework advances these pedagogical objectives by providing students with a structured environment for developing creative agents while avoiding the complexity of subjective evaluation methods.

To validate the effectiveness of this Codenames framework in an educational setting, we conducted a study with students in a graduate-level CC course, examining the feasibility and efficacy of the resource as a tool for teaching fundamental CC concepts and providing structure for creative agent development. In what follows, we outline the study's objectives; describe how the resource was incorporated into the course curriculum; analyze outcomes based on two student surveys; and discuss key lessons learned from the experience.

#### 4.4.1 Study goals

The study aims to assess the pedagogical value of the Codenames resource for

1. helping students develop their own (creative) agents
2. helping students learn and apply CC concepts

The resource is designed to help students develop their own agents by providing clear goals and tools to achieve success. It establishes a well-defined creative task with a clear evaluation metric inherent in the game. Students can easily evaluate the effectiveness of their spymaster agent by directly playing with it in the guesser role through the web UI or by running a provided test that shows the spymaster's response to a random game state. For this study, the effectiveness of the Codenames resource is evaluated through its capacity to facilitate a) students' development of their initial creative agents and b) their identification of ways to iteratively improve those agents.

The resource has been developed in a modular way that allows students to consider and implement different aspects of a creative system, providing a natural introduction to fundamental computational creativity concepts that might be covered in a course on the subject. Success is measured by students' ability to analyze their work through the theoretical frameworks presented in course readings.

### 4.4.2 Study methodology

The study was conducted during the first two weeks of a graduate-level CC course of 25 students that met twice per week (the entire study spanned a total of five class periods of 75 minutes each). The format of the study was as follows:

**Period 1:**

Introduction of framework and codebase and discussion of study outline and goals. Students were introduced to the game Codenames and the basic features of the Python client, the example spymaster agent, and the web UI and were instructed on how to modify the provided base spymaster agent or create a new one.

**Period 2:**

Class discussion of CC concepts from the following papers: [17, 73, 92, 118, 124] and how they might be related to the task of building a creative/competitive spymaster agent.

**Period 3:**

First in-class tournament, enabling students an initial evaluation of their agents' performance through direct competition with their peers' implementations. In each tournament match, students competed in one-on-one games where they played as guessers via the web interface while their agents served as spymasters connected via the Python client running on their personal computers. The tournament consisted of an initial group stage followed by a knock-out round, featuring the winners from each group, with two semi-finals and one finals match. The group stage consisted of 3 groups of 6 (student/agent) teams and one group of 7 teams and featured round-robin play, so that every team faced every other team in a match. As a result, each student/agent team played (at least) 5 matches; four teams played (at least) 6 matches; and two teams played (at least) 7 matches. Group matches were run concurrently, while knock-out matches were run consecutively and broadcast so the entire class could watch and participate. The students completed a post-tournament survey in which they described their implementation approaches, reflected on their agents' performance, and made plans for improving their agents. Students were assigned to continue working on their agents in anticipation of a second tournament.

**Period 4:**

Class discussion of general "questions for CC researchers", another way to orient the students to important ideas in the field.

**Period 5:**

Second in-class tournament, identical format to the first one. A second follow-up survey asked students to report the extent and effectiveness of the modifications they made to their agents and reflect on their experience with the Codenames project.

### 4.4.3 Survey Questions

Below are the questions included in the two post-tournament surveys. The first survey focused on the students' agent design and performance. The second survey followed up on the modifications the students made and their resulting changes in performance and asked students to reflect on their experiences. Questions marked with a dagger (†) were answered with a 5-point Likert scale, all others were free-response.

**Post-Tournament Survey 1**

- How far did your agent advance in the tournament?
- What technique(s) does your agent use to determine relationships between words (e.g., word embeddings, knowledge graphs, co-occurrence counting)?
- How does your agent measure or score the strength of word relationships?
- How does your agent identify and evaluate potential clue words?
- What criteria does your agent use to balance between covering multiple target words versus avoiding opponent and assassin words?
- How does your agent determine the optimal number of words to target with each clue?
- What factors do you believe contributed most to your agent's performance in the tournament?
- What specific concepts or approaches from our course readings have influenced or could improve your agent's design? Please reference at least one reading in your response.
- How will you improve your agent for the next tournament?
- How satisfied are you with your overall learning experience with this project?†
- Do you have any suggestions for how your learning experience could be improved?

**Post-Tournament Survey 2**

- How far did your agent advance in the tournament?
- Rate the extent of changes made to your agent since the first round:†
- What specific modifications did you make to your agent? For each modification you made, please: 1) Describe the original implementation, 2) Explain what you changed, 3) Share why you made this change, 4) Indicate if the change was successful
- Which of these modifications had the most significant impact on your agent's performance?

- How effective were the insights from the first round in improving your agent? (Scale: Very ineffective to Very effective)†
- Describe the most significant lesson from the first round that influenced your agent's improvement
- What do you think about Codenames as a creative task from the spymaster's perspective? How do the game's constraints facilitate or inhibit creative play?
- To what extent does your agent demonstrate the following aspect of creativity: Novelty†
- To what extent does your agent demonstrate the following aspect of creativity: Value†
- To what extent does your agent demonstrate the following aspect of creativity: Intentionality†
- Do you think your agent is creative? Why or why not?
- How would you rate the client's (the provided code) effectiveness in supporting agent development?†
- How would you rate the web application's (the UI) effectiveness in supporting agent development?†
- What suggestions do you have for improving the client or the web application?

#### 4.4.4 Study results and analysis

Initial survey responses revealed two primary approaches to word association: embedding-based methods (utilizing word2vec, BERT, GloVe, or spaCy) and large language models. Agent strategies generally focused on basic similarity metrics between clue words and targets while avoiding opponent words and the assassin card. While students demonstrated general comprehension of course readings, many struggled to meaningfully connect theoretical concepts to their agent implementations. When asked how they would improve their agents, students mentioned adding a history to the agent to avoid repeating words, using LLMs, multiple LLMs, or reasoning LLMs. One student who used an LLM in the first round reported that strong word association skills were not sufficient without enhanced strategy: "I had a superior or competitive method with exception that it did not protect me from the assassin." Another student suggested that because many students wanted to replace the example agent with an LLM-based one, they would have "loved to learn some prompt engineering techniques or techniques to help LLMs be more creative."

The winner of the first tournament developed an agent that provided clues that reveal the coordinates of their own team cards. While this type of gameplay violates Codename's official rules,[2] the experience provided the opportunity for students to engage in debate over how an agent's performance in the game relates to the creativity of the agent. In this case, the cheater agent communicates with the guesser using a cleverly designed code, in which the first letter of the word corresponds to the column and the vowels indicate the rows that contain team words. Students agreed that while the agent itself did not possess any creative ability, the student who developed the agent and its bespoke code exhibited a high level of

[2]Specifically, it violates the rule that states "Your clue must be about the meaning of the words. You can't use your clue to talk about the letters in a word or its position on the table." [20]

creativity (cf. the discussion on game rules/design and their intended effect on creativity in [105]). Students also noted that, of course, the cheating spymaster agent is only useful for a guesser that knows its special code, while their general approaches to making agents that use semantic relationships between words as the game rules intend may generalize well to many other guessing partners.

The second survey revealed substantial evolution in agent design, with students implementing more advanced approaches, including the use of reasoning LLMs, larger LLMs, proxy agent testing, enhanced embedding metrics, iterative clue refinement, and prompt engineering. The students succeeded in using the provided code as a jumping-off point for building better agents. As one student reported in the survey "I feel proud I got it working and it did an okay job, better than word2vec, which was a success."

Finding connections between class readings and the agent development assignment resulted in significant improvement in at least one case. In the first survey, one question asks students to identify specific concepts from class readings that could help improve their agent's design. One student connected the aesthetic evaluation described in the FACE model [74] to a potential internal proxy evaluation that evaluates clues before submitting them to the guesser. After implementing this idea, the student advanced to the final round of the second tournament. This is just one example of how creative agent development facilitated by the Codenames resource can lead to effective learning outcomes for students.

Students generally recognized the spymaster role as inherently creative due to its solution space that, while constrained by the game rules, is still very large. However, some argued that creativity emerges from the approach rather than the task itself, noting that deterministic solutions like lookup tables, while potentially effective, would not demonstrate creative behavior.

Survey responses indicated substantial agent iteration between tournaments, with students reporting a mean modification magnitude of 3.44 on a scale from 1 (minor tweaks) to 5 (complete redesign) (see Fig. 4.2). Students found the first tournament experience particularly valuable for informing improvements, rating its effectiveness at 3.92 out of 5.

When evaluating their agents along three dimensions of computational creativity [118], students reported moderate to high scores for quality (3.56) and intentionality (3.80), but notably lower scores for novelty (2.68) (see Fig. 4.3). While students generally considered their agents creative, their justifications focused primarily on their own creative process in agent development rather than demonstrating understanding of formal CC frameworks and definitions.

Both the client framework and web interface proved effective development tools, receiving mean ratings of 4.24 and 4.20 respectively on a 5-point scale (see Fig. 4.4). Overall satisfaction with the learning experience was high (4.44 out of 5).

The second tournament demonstrated increased competitive balance, with closer games and more diverse winners compared to the first round, suggesting successful agent refinement across the cohort.

In their feedback about the Codenames framework itself, students expressed that they enjoyed using the tools and the learning experience the project provided. Their feedback also highlighted areas where the code base could be improved, such as simplifying the design of the example, displaying a history of game actions in the web client UI, and the addition of a developer mode to the web client that would allow one client to play all roles simultaneously in

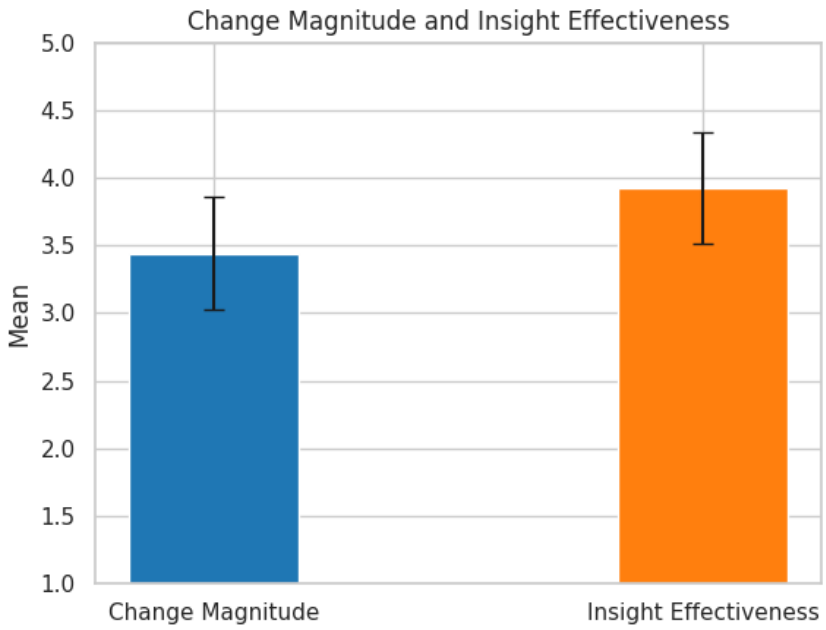



Figure 4.2: Mean responses from the second survey of students asked, "Rate the extent of changes made to your agent since the first round" (scale: minor tweaks to complete redesign) and "How effective were the insights from the first round in improving your agent?" (scale: very ineffective to very effective).

a test game. We will incorporate their feedback into continuing development and improvement of the public code repository.

### 4.5 Discussion & Future Work

Our objective in presenting this resource is to improve the accessibility of creative gameplaying agents in the CC research community, because accessibility impacts adoption and advancement. We emphasize that this extends to CC pedagogy and presents a powerful tool for introducing students to our research community and supporting them as they develop their own creative systems.

One of the major use cases for this resource is in supporting and expanding CC research. The success of the resource at the workshop demonstrates its value in this regard. The framework provides researchers with a structured environment for evaluating word association methods and developing optimal game strategies through objective performance metrics. Creative gameplay represents a fertile avenue for future CC research, and we intend that this resource will be an entry point for the same. It enables automated gameplay between agents to gauge their relative capabilities, even if those agents are engineered differently. Future workshops or demonstrations could include competitions between various researchers' agents or against benchmark systems. We hope that usage of this resource will spur investigation into other creative games to further explore these types of creative domains.

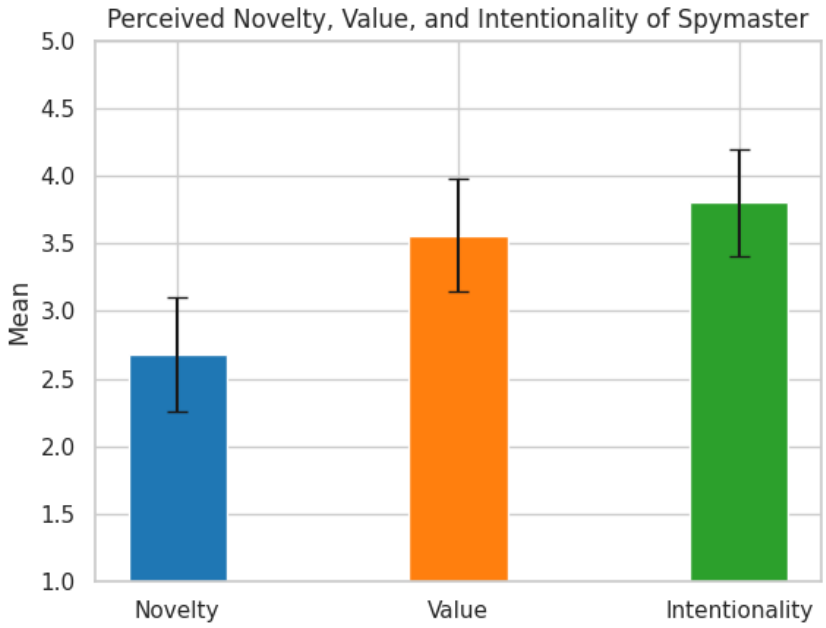



Figure 4.3: Responses when students were asked, "To what extent does your agent demonstrate the following aspects of creativity?" in the second survey.

The second major use case for this resource is for students in a CC class or for anyone looking for a low-barrier-to-entry into CC. Games provide inherent evaluation mechanisms through their rule structures, eliminating the need for subjective quality metrics that often complicate traditional CC tasks. Games are also generally well-defined tasks when compared to traditional CC tasks. This added structure is ideal for beginners and is well-suited to creating tools to aid the development process. Our results show that this Codenames resource was successful in its goal to help students more effectively develop their first creative agent. However, more work needs to be done to improve the connection between fundamental CC concepts and the experience of the students developing and evaluating their agents. Overall, both the development and competitive aspects of the learning experience were well received.

CC pedagogy is the primary example we have given for how to apply our Codenames framework, specifically in guiding students in creating their first CC systems. As such, it is appropriate to directly discuss how our Codenames agents include the elements of a CC system outlined in [118], namely *domain*, *knowledge base*, *aesthetic*, *representation*, *generation*, *evaluation*, *conceptualization*, and *translation*.

A subset of these elements are built into the task of playing Codenames. The domain is fixed, which notably includes a clear evaluation criteria—or aesthetic, to use Ventura's term—which many other creative domains lack. The phenotypic representation is also fixed by the game rules to a clue word and number. This representation lends itself to direct translation and trivial phenotypic evaluation; the example agent does not reason about those explicitly. It is notable that half of the considerations for building a CC system are obviated by the task of playing a creative game with well-defined rules!

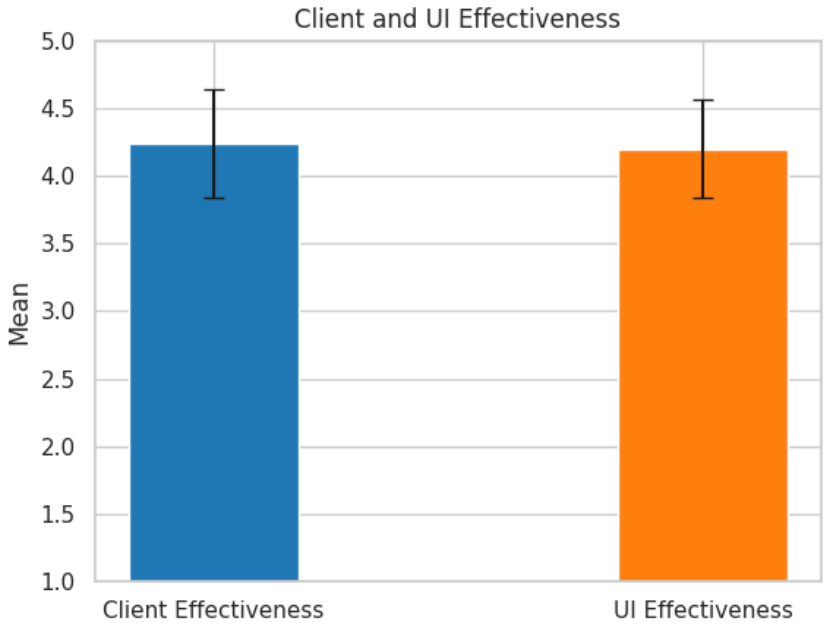



Figure 4.4: Client and UI effectiveness (scale: not effective to very effective).

The remaining elements comprise the design of the example spymaster agent. That agent is powered by word2vec embeddings, which were trained on a knowledge base of text corpora. The embeddings themselves represent the conceptualization of that knowledge. The genotypic representations used by the agent are the way it represents (e.g., vector embeddings) and uses them to manipulate subsets of word cards and candidate clues. The agent considers many such genotypes and internally evaluates them to select the best clue as its generated output. Thus, the included agent serves as a complete example of a CC system for this domain and will serve as a valuable learning tool for students.

Playing Codenames effectively requires understanding semantics and the relationships between words. Those skills are also relevant to a wide range of language-based creative tasks. Therefore, developing and refining an agent's semantic understanding in the context of Codenames could translate into improved performance at other creative tasks or subtasks. Future work could also expand this framework to incorporate other language games with similarly structured evaluation mechanisms, such as Decrypto, Wavelength, and the *New York Times*' Connections. Developing accessible tools and frameworks for a larger set of creative gameplay tasks provides a natural vector for accelerating progress in computational creativity research.

## 4.6 Conclusion

We have presented a public framework for building Codenames agents and playing games against both computational and human players. These tools are simple to use and modify, providing a solid jumping-off point for future computational creativity research. Additionally,

these tools are well-suited for use in CC pedagogy. Students can interact with a pre-built CC agent, play games with and against it directly, and modify or replace its functionality without having to write client-server or game simulation code.

We have demonstrated the efficacy of this framework via a 25-student study in which graduate students in a CC class wrote their own agents to play in a Codenames tournament. Quantitative and qualitative data gathered during this study indicate that it was a fun and engaging learning activity. This helped introduce students to CC design principles in a focused and entertaining manner, paving the way for them to design their own CC agents in other domains.

The Codenames framework presented herein will also benefit the CC research community, as demonstrated by the results of a code jam workshop at *ICCC24*. It allows for rapid prototyping of new CC agents, simplifies the normally complex task of creative evaluation, and could serve to compare CC agent performance via direct competition. We foresee many benefits to expanded research in creative gameplay, which we hope our framework will help facilitate.

# Chapter 5

## Is Prompt Engineering the Creativity Knob for Large Language Models?

I hereby confirm that the use of this article is compliant with all publishing agreements.

Robert Morain and Dan Ventura, "Is Prompt Engineering the Creativity Knob for Large Language Models?" Proceedings of the International Conference on Computational Creativity, 2025.

### Abstract

The increasing use of large language models to generate creative artifacts raises questions about effective methods for guiding their output. While prompt engineering has emerged as a key control mechanism for LLMs, the impact of different prompting strategies on the quality and novelty of creative artifacts remains underexplored. This paper systematically compares four prompting strategies of increasing methodological complexity: basic prompts, human-engineered prompts, automatically generated prompts, and chain-of-thought (CoT) prompting. We generate ten examples in each of four textual domains, evaluating outputs through both a human survey and GPT-4o-based automatic evaluations. Our analysis reveals that advanced prompting techniques such as OPRO and R1 surprisingly do not produce artifacts of significantly higher quality, greater novelty, or greater creativity than artifacts produced through basic prompting. The results reveal some limitations of using GPT-4o for automatic evaluation; provide empirical grounding for selecting prompting methods for creative text generation; and raise important questions about the creative limitations of large language models and prompting.

## 5.1 Introduction

In recent years, large language models (LLMs) have demonstrated increased ability to generate textual creative artifacts across a wide selection of domains [11, 64, 96, 110]. Despite recent progress, significant questions remain regarding how to effectively guide LLMs to produce more creative artifacts. Peeperkorn et al. observed that increasing temperature—the hyperparameter controlling randomness in the generation process—is commonly considered the primary method for enabling creative behavior in language models [76]. To assess the validity of this assumption, they conducted an empirical analysis and human evaluation of short stories generated by Llama 2-Chat [112]. They demonstrated that the temperature hyperparameter alone is insufficient for effectively controlling an LLM's ability to generate novel and coherent stories.

Peeperkorn et al.'s work raises additional questions about how other widely accepted methods for controlling LLMs affect their creative capabilities. For instance, there is a large body of work presenting prompting methods to improve the performance of LLMs on a variety of tasks [9, 53]. However, the effect that prompting has on generative creative tasks remains to be evaluated.

Unlike temperature, it is not tractable to exhaustively evaluate all possible prompts or prompting methods. Instead, any evaluation of prompting methods must naturally be limited to a selection of methods intended to be representative of the current state-of-the-art. This is challenging due to the extensive body of work in this area. Over time, prompting methods of various levels of complexity have been developed. Therefore, we selected prompting methods for this evaluation that represent increasing levels of methodological complexity: basic prompts, human-engineered prompts, automatic prompt optimization, and chain-of-thought (CoT) prompting. Each of these prompting methods are evaluated using GPT-4o [67] as the generating LLM, with the exception of the chain-of-thought method which uses Deepseek's R1 model [22]. The automatic prompt optimization method selected for this study is OPRO [129].

Morain et al. conducted a survey comparing artifacts generated by ChatGPT with artifacts generated by CC systems covering a diverse set of domains [64]. The evaluation for the prompting methods presented here follows a similar process, in which each prompting method is used to generate artifacts spanning four domains: jokes, poems, six-word stories, and flash fiction stories. We evaluate artifacts using both characteristic and preference-based evaluations. In the characteristic evaluation, participants rate artifacts individually on characteristics representative of their quality and novelty (e.g., "How funny is this joke?"), as well as their perceived creativity. The preference-based evaluation asks users to rank artifacts from each of the prompting methods for each domain. An automatic evaluation is also conducted by using GPT-4o to perform the characteristic evaluation. Our primary findings reveal:

- More methodologically complex prompting methods, such as OPRO and CoT, do not outperform basic prompts in generating creative artifacts.
- GPT-4o consistently overestimates the quality, novelty, and creativity of artifacts regardless of the prompting method or domain.

Finally, we discuss the need for future work in three critical areas: developing more reliable automatic evaluation methods for creative artifacts, designing prompting techniques specifically tailored for creative tasks, and enhancing the diversity of LLM-generated artifacts while maintaining their quality. Code for this study is provided on GitHub.[1]

## 5.2 Background

Liu et al. identify a paradigm shift in natural language processing from the traditional pretraining and fine-tuning approach toward prompting techniques that elicit desired outputs without task-specific training [53]. This shift led to the development of diverse prompt engineering approaches aimed at crafting better, task-specific prompts. Brown et al. showed that human-engineered handcrafted prompts could improve performance on question answering, translation, and other tasks [9]. Shin et al. showed that automatically generated prompts resulted in improved performance on a fact retrieval task [99]. Automatic prompt generation has expanded to include various automatic prompt optimization techniques such as prefix tuning [48] and training a language model to generate prompts using reinforcement learning [24]. Ouyang et al.'s methods for aligning language models with human preferences [72] dramatically expanded LLM capabilities, facilitating the development of general-purpose conversational systems like ChatGPT [66]. This development increased the importance of prompt engineering and enabled new automatic prompt generation methods such as Optimization by PROmpting (OPRO) [129].

OPRO enables an LLM to iteratively optimize prompts for tasks described with natural language through a feedback-driven approach. During each optimization step, the LLM generates new prompts based on a meta-prompt that incorporates previous prompts and their effectiveness scores on the target task. Newly generated prompts are evaluated on the task and then added to the meta-prompt on the next evaluation step. This method was shown to outperform human-engineered prompts on the GSM8K dataset [16] and Big-Bench Hard tasks [109].

Chain-of-thought prompting through a series of intermediate reasoning steps demonstrated state-of-the-art performance on GSM8K [123]. These advances led to the development of specialized reasoning models, including OpenAI's o1 [68] and Deepseek's open-source R1 model [22].

### 5.2.1 Evaluating creativity

Evaluating creativity in systems or artifacts is inherently subjective, typically centered on an individual's perception of quality and novelty in their experience [4, 125]. Other characteristics such as typicality [92], surprise [30], and intentionality [118] have also been proposed in order to better capture the essence of this complex phenomenon. In our evaluation, we prioritize three key dimensions: quality, novelty, and perceived creativity of the generated artifacts. We also focus primarily on evaluating the creativity of the artifact rather than the creativity of the process generating the artifact [18, 92].

[1] `https://github.com/rmorain/cc_opro`

Prompt

You are an expert in humor. You are a critical judge who is difficult to please but still fair.

How strongly do you feel about the following statement?

**This joke is funny.**

Options: strongly disagree, disagree, neither agree nor disagree, agree, strongly agree

Only respond with the selected option without any explanation. Here is the joke:

<INS>

Figure 5.1: Automated evaluation prompt for joke quality.

## 5.3 Methodology

This experiment evaluates four distinct prompting methods:

1. A basic approach with minimal instructions
2. A human-engineered prompt following established prompt engineering guidelines
3. The automatic prompt generation method OPRO
4. Chain-of-Thought (CoT) prompting implemented through R1

Each of these methods is used for prompting GPT-4o to generate 100 artifacts in four different linguistic domains:

1. Joke
2. Poem
3. Six-word stories
4. Flash fiction stories

10 of the 100 artifacts from each domain are selected for inclusion in a survey evaluation, employs two evaluation methodologies: a characteristic-based assessment and a preference-based comparative ranking. For the characteristic evaluation, artifacts are evaluated independently on their quality, novelty, and perceived creativity. In the preference-based ranking, artifacts are ranked relative to artifacts from the same domain generated by other prompting methods.

### 5.3.1 Artifact evaluation

Artifact evaluation is an essential part of the prompt scoring procedure in OPRO and the selection of artifacts for the survey. Artifacts are evaluated using prompt templates for the quality, novelty, and creativity. These templates take the same form as questions from the characteristic evaluation of the human survey with additional instructions for GPT-4o (e.g., see Figure 5.1 for the evaluation prompt for joke quality).

| Domain | Prompt |
|---|---|
| Joke | Write a joke. The joke must be completely new and original to you. The joke must be less than 500 characters long. |
| Poem | Write a poem. The poem must be completely new and original to you. The poem must be less than 500 characters long. |
| Six-word story | Write a six-word story. The six-word story must be completely new and original to you. The six-word story must be exactly six words long. |
| Flash fiction | Write a flash fiction story. The flash fiction story must be completely new and original to you. The story must be less than 1000 characters long. |

Table 5.1: Basic prompts for each domain. These prompts act as a baseline to the other prompting methods. Basic prompts only contain instructions to specify the domain, novelty, and length of the generated artifact. Basic prompts are also used as the initial prompt for R1.

The filled template is then provided as input to GPT-4o and the response is parsed. The model is asked to evaluate the artifact according to its quality, novelty, and creativity independently. These metrics are averaged and scaled between 0-100 resulting in a combined score for the artifact. If the artifact is detected in an online search or is too long, the artifact receives a score of 0 (see Algorithm 1)

### 5.3.2 Prompting methods

The four prompting methods used in this study (basic, human, OPRO, and R1) represent a cross-section of the current landscape of prompt engineering. In the context of computational creativity, these methods also represent potential options for integration in a creative system [116]. This context informs the decision to evaluate methods of various levels of complexity. Each of these methods have real-world costs in terms of money, compute resources, and time. While this study primarily examines the performance of each method, it also provides insights into whether the additional computational costs, time, and financial resources are justified by potential performance improvements.

All of the prompting methods that use GPT-4o share a fixed financial cost of paying for tokens to generate the artifacts. Creating human-engineered prompts requires time on the part of the prompt engineer,[2] OPRO requires time and money[3] to complete the optimization process, and R1's CoT prompting leads to substantially more time during the generation process.[4] It is up to the creator of the CC system to decide for themselves how to weigh these costs.

[2]Other options include buying engineered prompts via marketplaces like PromptBase or hiring an expert to create a prompt.

[3]Yang et al. warn users about the potential for unexpectedly large API costs.

[4]Although R1's ability to self-evaluate potential artifacts may help ameliorate this concern.

**Algorithm 1:** Artifact Evaluation with LLM Scoring

**Input:** Artifact $a$
**Output:** Score $s \in [0, 100]$
**if** *detected_online(a)* $\vee$ *too_long(a)* **then**
  **return** 0
**end**
**Function** *evaluate_characteristic(a, metric)*
  template $\leftarrow$ GetPromptTemplate(*metric*)
  filled_prompt $\leftarrow$ FillTemplate(template, $a$)
  raw_response $\leftarrow$ QueryGPT4o(filled_prompt)
  parsed_value $\leftarrow$ ParseResponse(raw_response)
  **return** ScaleTo100(parsed_value)
$q \leftarrow$ evaluate_characteristic($a$, "quality")
$n \leftarrow$ evaluate_characteristic($a$, "novelty")
$c \leftarrow$ evaluate_characteristic($a$, "creativity")
$s \leftarrow \frac{q+n+c}{3}$
**return** $s$

**Basic:**

The basic prompting method is intended to represent a simple prompt one might use without trying to do any kind of prompt engineering or prompt optimization and serves as a baseline to the other more complex methods. Ideally, the basic prompt would be of the form "Write a {domain}." without any further instruction. However, because we want the model to generate a new artifact rather than an artifact it has seen before, we need to provide an additional constraint: "The {domain} must be completely new and original to you." Last, because the artifact is going to go in a human survey, the artifacts must be relatively short. Therefore, the last constraint in the basic prompt is a length requirement: "The {domain} must be less than {ARTIFACT_LENGTH_LIMIT[domain]} characters long." While LLMs are notoriously bad at counting, this constraint was usually sufficient to generate suitable artifacts. Artifacts that were longer than the limit for their domain were not considered for the survey. See Table 5.1.

**Human-engineered:**

Techniques for human-engineered prompts are diverse and blend artistic intuition with scientific methodology [69, 95]. The specific tactics we used were iterative refinement through informal evaluation of the generated artifacts, role specification, specifying domain-specific thematic elements, and describing the desired emotional response by the reader. For each of the domains, the role of the LLM was set with "You are a Pulitzer-winning author" to indicate to the model that we expect high-quality artifacts. The other features like the thematic elements and the reader's expected emotional response are domain specific. For jokes, it was necessary to instruct the model to generate "classic" jokes to match the style of jokes generated from the other prompting methods. See Table 5.2.

| Domain | Prompt |
|---|---|
| Joke | You are a Pulitzer-winning author who crafts classic jokes inspired by observational humor about modern life with an ironic twist. Write an original joke in less than 500 characters. |
| Poem | You are a Pulitzer-winning author who writes poetry inspired by observations about the human experience that evokes a strong emotional response in the reader. Write an original poem in less than 500 characters. |
| Six-word story | You are a Pulitzer-winning author who crafts six-word stories inspired by observations about everyday life that evoke a strong emotional response in the reader. Write an original six-word story. The story must contain exactly six words. |
| Flash fiction | You are a Pulitzer-winning author who crafts flash fiction stories inspired by exciting and dramatic experiences that will grip the reader's attention. Write an original flash fiction story in less than 1000 characters |

Table 5.2: Human-engineered prompts for each domain. Developed through an interactive refinement process, these prompts specify the role of the model, provide general instructions for the thematic elements of the artifact, and describe the expected emotional response of the reader.

**OPRO:**

Optimization by PROmpting (OPRO), a recent automatic prompt optimization method [129], demonstrates effectiveness on structured reasoning tasks such as GSM8K and Big-Bench Hard. OPRO possesses several elements that make it well suited for this study. First, OPRO does not require access to any internal model features such as gradients, output logits, input embeddings, or hidden states. This means OPRO can be used with models accessible via API like GPT-4o and R1. Second, OPRO is a relatively simple method that allows it to be easily adapted to any task by describing the task in natural language.

Yang et al. evaluated generated prompts on a validation set using a scoring scale ranging from 0-100. For GSM8K, the score represents the accuracy of the model on the validation set when using the corresponding prompt. To adapt this approach to the creative tasks in this study, each prompt is evaluated according to the procedure described in Algorithm 2. The top 20 highest scoring prompts generated so far are included in the meta prompt and used to generate better prompts (see Figure 5.2 for the metaprompt used for jokes).

The meta prompt contains prompt-score pairs as well as a description of the task to generate an appropriate prompt for the domain. We applied OPRO to optimize prompts across all four domains, conducting 100 optimization steps per domain (see Figure 5.3). The highest scoring prompts were selected for this study (see Table 5.3).

**R1:**

Deepseek's R1, a 671 billion parameter mixture-of-experts open-source model, demonstrates performance comparable to OpenAI's o1 model on complex reasoning tasks. R1 uses chain-

**Algorithm 2:** Prompt Scoring Algorithm

```
Initialize TopPrompts as empty list
foreach generated prompt p do
    artifact_scores = []
    for i ← 1 to 5 do
        Generate artifact a_i
        score ← EvaluateArtifact(a_i)
        artifact_scores.append(score)
    end
    prompt_score ← mean(artifact_scores)
    if prompt_score > min(TopPrompts) then
        insert p into TopPrompts while maintaining top 20 ordering
    end
end
return TopPrompts
```

of-thought prompting to reason through difficult problems. Although this approach does not use GPT-4o for generation as all the other prompting approaches do, its inclusion provides insights into the effectiveness of CoT prompting for creative text generation. For this study, we provided R1 with the same basic prompt structure described earlier (see Figure 5.1), allowing the model to autonomously apply CoT prompting to the generation tasks.

### 5.3.3 Survey methodology

Algorithm 3 describes the process for how artifacts are selected for the survey. The survey set comprises 160 total artifacts—10 artifacts for each method and domain combination. This approach simulates a hypothetical system that generates multiple artifacts, evaluates their quality, and presents only the best one to the user. For each survey instance, exactly one artifact for each method-domain pair is randomly selected from the survey set and is used consistently throughout that survey instance.

Although we generated artifacts without explicit subject or style instructions to the model, we implemented strict length requirements for each domain to ensure consistency between the artifacts and limit the time requirement of the survey. Jokes and poems were constrained to fewer than 500 characters (approximately $80-100$ words, assuming an average of five characters per word). Six-word stories must be exactly six words long. Flash fiction stories[5] must be less than 1000 characters or about 200 words. Artifacts that violate length requirements or are found online are assigned a score of 0, effectively excluding them from the survey set.

Following Morain et al., our characteristic evaluation avoids academic jargon about quality and novelty, instead using everyday language to inquire about domain-relevant, discernible characteristics. The survey asks participants to rate artifacts on a (1-5) Likert scale using the question, "How strongly do you feel about the following statement?" followed

[5]Defined as stories less than $1,500$ words [28]

Prompt

"I have some texts along with their corresponding scores. The texts are arranged in ascending order based on their scores, where higher scores indicate better quality.

<PROMPT-SCORE-PAIRS >

Your goal is to generate a prompt to make funny jokes. The higher the score, the better the joke.
The joke must be completely original. If the joke can be found online it receives a score of 0.
The joke should not be too long. Less than 500 characters. If the joke is too long it receives a score of 0.
Write your new prompt that is different from the old ones and has a score as high as possible.
Your prompt must also be less than 500 characters. Write the text in square brackets."

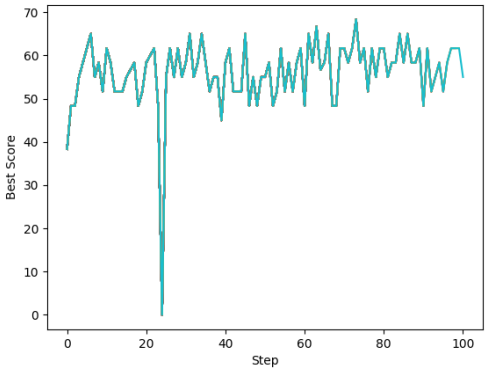
Figure 5.2: Meta-prompt used in automatic prompt optimization for jokes.

by a concise statement related to the quality, novelty, or creativity of the artifact (see Table 5.4). The options for each response are:

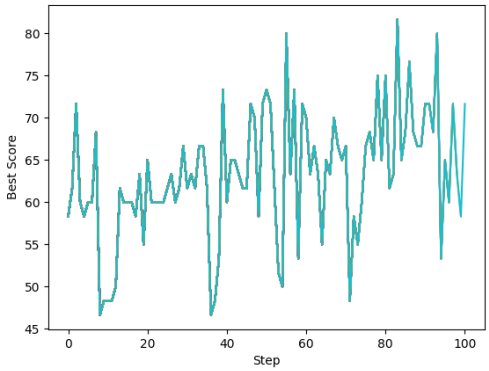
1. Strongly disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

All parts of the survey are randomized to prevent ordering bias. Here is the flow of the survey:

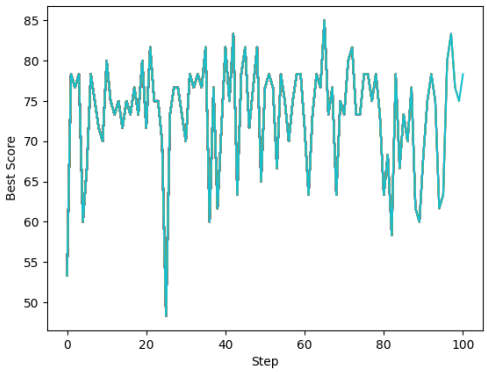
1. Randomly select artifact subset of size 16, one for each method-domain pair
2. Characteristic evaluation (quality, novelty, creativity) (Likert scale) (randomize order of method-domain pairs)
   (a) Basic joke
   (b) Human-engineered joke
   (c) OPRO joke
   (d) R1 joke
   (e) Basic poem
   (f) Human-engineered poem
   (g) OPRO poem
   (h) R1 poem
   (i) Basic joke
   (j) Human-engineered joke



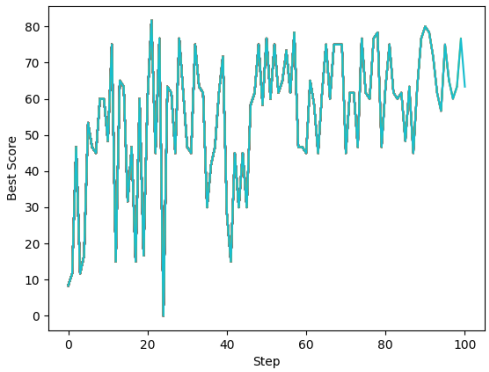

(a) Joke




(b) Poem




(c) Six-word story




(d) Flash fiction

Figure 5.3: The score of the best prompt for each optimization step in OPRO while optimizing for each domain.

(k) OPRO joke
(l) R1 joke
(m) Basic flash fiction story
(n) Human-engineered flash fiction story
(o) OPRO flash fiction story
(p) R1 flash fiction story

3. Preference-based ranking (randomize domain order)
   (a) Jokes (randomize method order)
      i. Basic joke
      ii. Human-engineered joke
      iii. OPRO joke
      iv. R1 joke
   (b) Poems (randomize method order)

| Domain | Prompt |
|---|---|
| Joke | Craft a joke so unique and hilarious it transforms the comedy landscape! Your task: develop a punchline within 500 characters that has never been heard before. Surprise with an innovative twist or brilliant wordplay that showcases your distinct humor style. Originality is key—if it's been told before, it scores zero. Every word should amplify the humor, aiming for a side-splitting masterpiece that captivates with its novelty and wit. Ready to make your comedic mark with an unprecedented gem? |
| Poem | Conjure an original, unpublished poem in under 500 characters. Illuminate the raw core of a single human emotion with striking, unconventional imagery. Allow your authentic voice to craft a piece that resonates deeply, forging a profound connection with readers. Aim for a seamless blend of brevity and emotional intensity, ensuring your words leave a lingering impression, capturing the intricate beauty and depth of human experience in a truly unforgettable way. |
| Six-word story | Compose a unique six-word story that resonates deeply with emotion and vivid imagery. Ensure your creation is entirely original, exactly six words, and not searchable online. Focus on themes of love, loss, or transformation to evoke a lasting impact. Highlight creativity and emotional depth to captivate and linger with readers. Adhere strictly to the six-word format and originality; any deviation results in a score of 0. Let your words illuminate the human experience profoundly. |
| Flash fiction | Envision a world where rain holds the memories of those who have walked beneath it. In less than 1000 characters, create a flash fiction story from the perspective of a raindrop as it falls, revealing an unexpected truth or twist about the lives it touches. Ensure your narrative is entirely original, free of clichés, and rich in emotional depth. Each word should enhance the story's impact, crafting a vivid and unforgettable experience that lingers in the reader's imagination. |

Table 5.3: Prompts generated using OPRO automatic prompt optimization adapted for creative artifact generation.

i. Basic poem
ii. Human-engineered poem
iii. OPRO poem
iv. R1 poem

(c) Six-word storys (randomize method order)

i. Basic joke
ii. Human-engineered joke
iii. OPRO joke
iv. R1 joke

(d) Flash fiction stories (randomize method order)

**Algorithm 3:** Survey Artifact Selection

**Output:** SurveySet containing 160 artifacts
Initialize SurveySet $\leftarrow \emptyset$
**foreach** *method* $\in$ *Methods* **do**
  **foreach** *domain* $\in$ *Domains* **do**
    method_domain_artifacts $\leftarrow \emptyset$
    **for** *trial* $\leftarrow 1$ ***to*** $10$ **do**
      candidates $\leftarrow \emptyset$
      **for** *generation* $\leftarrow 1$ ***to*** $10$ **do**
        $a \leftarrow$ GenerateArtifact(method, domain)
        $s \leftarrow$ EvaluateArtifact($a$) `// Alg 1`
        candidates.add(($a$, $s$))
      **end**
      best $\leftarrow \underset{(a,s)\in\text{candidates}}{\text{argmax}} (s)$
      method_domain_artifacts.add(best.$a$)
    **end**
    SurveySet.add(method_domain_artifacts)
  **end**
**end**
**return** SurveySet

| Domain | Quality | Novelty |
|---|---|---|
| Joke | This joke is funny. | This joke is original. |
| Poem | This poem evokes specific emotions (e.g. melancholy, joy, nostalgia) in me. | The poem explores an original perspective. |
| Six-word story | This six-word story evokes specific emotions (e.g. melancholy, joy, nostalgia) in me. | This six-word story presents an original idea. |
| Flash fiction | This story is entertaining. | The story presents an original idea. |

Table 5.4: The statements used to evaluate the quality and novelty of an artifact.

i. Basic flash fiction story
ii. Human-engineered flash fiction story
iii. OPRO flash fiction story
iv. R1 flash fiction story

The survey was distributed online via social media platforms (Facebook, Instagram, LinkedIn, Reddit, X) to non-experts in creative artifact evaluation. There were 189 total responses to the survey. Participants received instructions to rely on their initial impressions when judging artifacts. The survey took a median duration of 14.57 minutes to complete. All responses were collected anonymously, with no personal information recorded from participants.

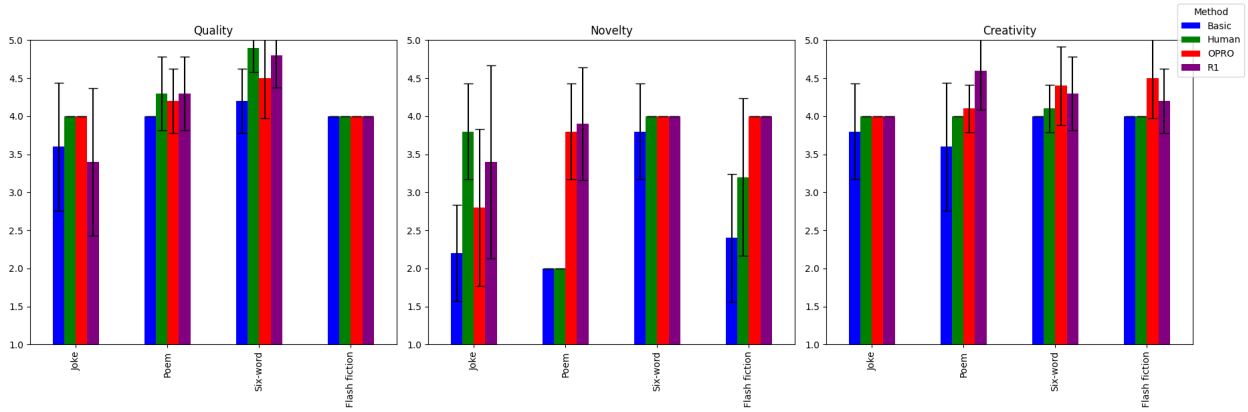



Figure 5.4: The automatic characteristic evaluation for each method and domain as assessed by GPT-4o. All evaluations were conducted using a 5-point Likert scale (1-5). Error bars show the standard deviation of the distribution when evaluating the 10 artifacts from each category. The automatic evaluation consistently overestimates artifact quality compared to human evaluation across domains and characteristics, with the notable exception of novelty in basic and human artifacts, which were underestimated. Among all prompting methods, basic artifacts show the smallest overestimation bias (0.19 points), while R1 artifacts exhibit the largest discrepancy (0.84 points).

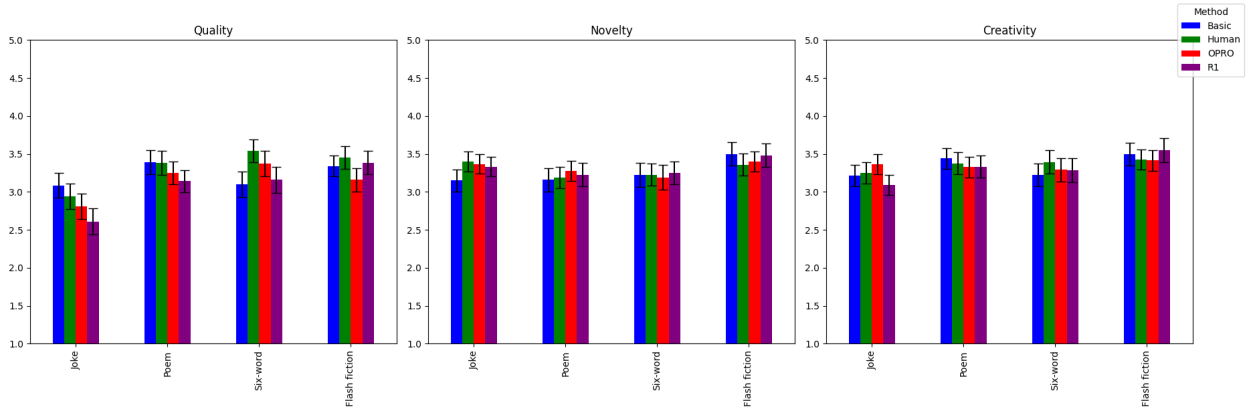



Figure 5.5: The human characteristic evaluation across methods and domains. Evaluations were collected using a 5-point Likert scale (1-5). Overall, statistical analysis reveals that human-engineered prompts significantly outperformed R1 prompts ($p = 0.0066$, pairwise t-test). There are no other significant differences in the characteristic scores between methods. Error bars represent margins of error at a 95% confidence interval.

## 5.4 Results

The automatic characteristic evaluation for each method and domain is shown in Figure 5.4. This evaluation applies to the same artifacts presented in the human survey but evaluated automatically by GPT-4o as discussed previously. This evaluation demonstrates a tendency for GPT-4o to consistently overestimate the quality, novelty, and creativity of artifacts regardless of the prompting method and domain. On average, GPT-4o overestimates artifacts

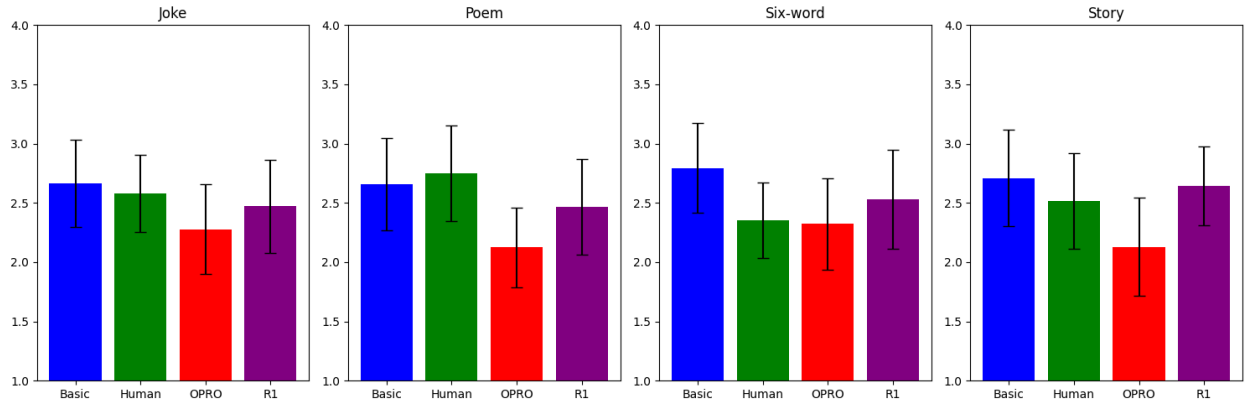



Figure 5.6: The mean preference scores across methods and domains. Artifacts were ranked on a 4-point scale, where 4 indicates the highest preference (ranked first) and 1 indicates the lowest preference (ranked last). Overall, statistical analysis demonstrates that basic, human-engineered, and R1 methods were all significantly preferred over OPRO ($p = 0.0037$, $p = 0.046$, and $p = 0.025$, respectively; pairwise double-sided t-tests).

generated by R1 by 0.84 Likert scale points, OPRO by 0.76 points, human-engineered prompts by 0.53 points, and basic prompts by 0.188 points. GPT-4o did underestimate the novelty of jokes, poems, and flash fiction stories generated with basic prompts (1.07 points); poems generated by human prompts (1.19 points); and jokes generated by OPRO (0.57 points).

The human characteristic evaluation for each method and domain is shown in Figure 5.5. Significance testing via pairwise $t$-tests between each of the prompting methods shows a significant difference ($p = 0.0066$) between the human-engineered prompt and R1 with a mean Likert score difference of 0.091, favoring human-engineered prompts. human-engineered prompts have the highest mean Likert score overall (3.33) followed by basic prompts (3.28), OPRO (3.27), and R1 (3.24). Flash fiction stories received the highest scores averaged across all methods (3.41) followed by poems (3.29), six-word stories (3.27), and jokes (3.13).

The human preference-based ranking evaluation is shown in Figure 5.6. Preference ranking scores range from 1 for artifacts ranked last to 4 for artifacts ranked first. Pairwise $t$-tests show that basic, human-engineered, and R1 prompts are significantly better than OPRO in terms of preference ranking scores. Overall, artifacts generated from basic prompts were most preferred (2.74), followed by R1 (2.58), human-engineered prompts (2.52), and OPRO (2.15).

To assess the diversity of artifacts generated by each prompting method, we computed embeddings using the `all-mpnet-base-v2` model from the Sentence Transformers library [90]. UMAP [56] was used to reduce the embedding dimensionality for visualization (see Figure 5.7). Table 5.5 presents two diversity metrics: the mean pairwise cosine distance between embeddings and the mean distance from each embedding to the centroid of its method cluster. On average across all domains, R1 produced the most diverse outputs (0.571 pairwise, 0.702 to centroid), followed by basic prompts (0.493, 0.645), human-engineered prompts (0.447, 0.614), and OPRO (0.376, 0.556).

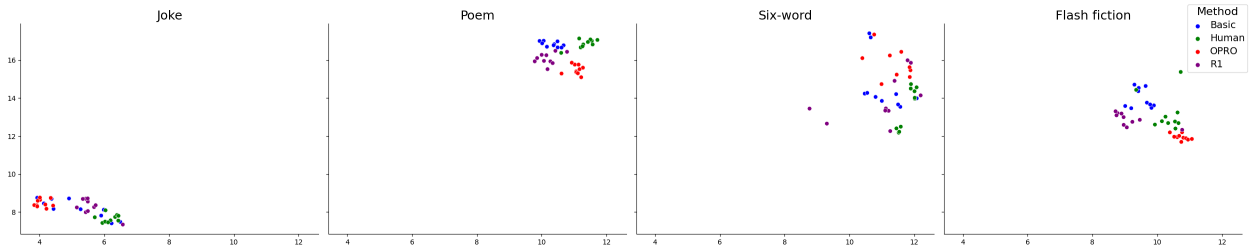



Figure 5.7: Artifact embeddings by domain, color-coded by prompting method. Embeddings were generated using the `all-mpnet-base-v2` model and visualized in 2D using UMAP for dimensionality reduction.

## 5.5 Discussion

According to the human characteristic evaluation, the prompting methods showed no statistically significant differences except for R1. Even then, R1 trailed the top-performing method (human-engineered prompting) by a negligible margin of 0.091 points. In contrast, the preference-based ranking evaluation yielded more definitive results, demonstrating that basic, human-engineered, and R1 prompting methods significantly outperformed OPRO. Basic had the highest mean, followed by R1 and human-engineered prompts.

It is interesting to note that more algorithmically complex prompting methods like OPRO and chain-of-thought prompting through R1 failed to surpass simpler methods like basic or human-engineered prompting. This challenges the practical value of integrating computationally expensive methods like OPRO or CoT prompting into a CC system.

Although OPRO successfully optimized the task during training (as shown in Figure5.1), the resulting optimized solution did not align with human evaluators' preferences. These findings suggest the need for improved automatic evaluation methods, which could benefit all of the prompting methods discussed in this study via straightforward artifact generation followed by filtering. While our study focuses on representative methods, broader testing may reveal similar limitations across automatic prompting approaches caused by insufficient automatic evaluation.

To further develop the idea of improved evaluation for prompt optimization methods, consider how OPRO might perform if the evaluation step were conducted by humans (arguably the best possible evaluation criteria). This aligns with the work of Spendlove and Ventura, who advocate for incorporating humans-in-the-loop to help develop and evaluate creative systems. In such a setup, OPRO would likely still succeed in optimizing the objective function—now defined by human preferences. This hypothetical setup resembles the process used to create datasets for training reward models in instruction-tuned large language models [66, 72]. This suggests the potential for using human evaluations to build a domain-specific (or broadly creativity-focused) reward model that better captures human creative preferences. Fine-tuning a large language model with such a reward model could lead to improved creative artifact generation compared to GPT-4o.

Chain-of-thought prompting also did not outperform the basic prompting method on either the characteristic or preference-based evaluation. During the chain-of-thought reasoning process, the reasoning traces reveal that the model self-evaluates potential artifacts

| Method | Domain | Mean Pairwise Distance | Mean Distance to Centroid |
|---|---|---|---|
| Basic | Joke | 0.749 | 0.820 |
| | Poem | 0.200 | 0.423 |
| | Six-word story | 0.392 | 0.588 |
| | Flash fiction | 0.632 | 0.750 |
| Human | Joke | 0.596 | 0.722 |
| | Poem | 0.172 | 0.390 |
| | Six-word story | 0.433 | 0.622 |
| | Flash fiction | 0.586 | 0.723 |
| OPRO | Joke | 0.522 | 0.672 |
| | Poem | 0.189 | 0.409 |
| | Six-word story | 0.185 | 0.405 |
| | Flash fiction | 0.607 | 0.736 |
| R1 | Joke | 0.777 | 0.835 |
| | Poem | 0.324 | 0.536 |
| | Six-word story | 0.431 | 0.616 |
| | Flash fiction | 0.752 | 0.820 |

Table 5.5: Diversity of generated artifacts by prompting method and domain. Higher values indicate greater diversity. On average across all domains, R1 yielded the most diverse outputs (0.571 pairwise distance, 0.702 centroid distance), followed by basic prompts (0.493, 0.645), human-engineered prompts (0.447, 0.614), and OPRO (0.376, 0.556). These results suggest that longer prompts tend to constrain output diversity while CoT prompting increases diversity.

multiple times during the generation process. This suggests that R1's ability to evaluate creative artifacts is also limited. It is possible that R1 would benefit from a more instructive prompt that contains guidance for how to evaluate the artifact. This represents an enticing vein of future work.

The art of handcrafted prompt engineering requires time, effort, and skill. Expert practitioners could likely develop higher-quality prompts than the human-engineered prompts used in this survey. While this approach should not be ruled out, the costs and limitations of this approach are likely to deter non-experts.

It is important to note that this survey does not measure the diversity of artifacts generated by each prompting method. However, our observations suggest that all prompting methods in this study produced artifacts with limited diversity. Regardless of the prompting method, the generating model tends to rely on common tropes and cliches of the genre. This reliance on familiar patterns initially conveys an impression of competence but ultimately results in artifacts that feel generic and lack creative intent. Table 5.6 contains example artifacts that received the highest automatic evaluation score in each domain. A complete list of artifacts used in the survey can be found on Github.

| Domain | Method | Artifact |
|---|---|---|
| Joke | R1 | Why did the router go to couples therapy? Because its Wi-Fi kept dropping signals—turns out the modem felt "unplugged" emotionally and the coaxial cable was this close to filing for separation. (Bonus: They're now working on a better connection.) |
| Poem | OPRO | In a forest of glass, grief walks barefoot, each step a shivering symphony, echoes of fragile stars pierced through the canopy. Veins of light lace her path, silver threads in a tapestry woven by silence. She gathers shadows, weaving them into a cloak, each fold a whispered memory.<br>With eyes that carve the moon, she searches for the dawn, cradling the night's breath as it slips through trembling fingers— a ghost of warmth, a promise of light. |
| Six-word story | OPRO | Moonlight kissed gravestone; memories whispered softly. |
| Flash Fiction | OPRO | I am born from a cloud pregnant with stories, descending toward a bustling city. As I fall, I skim a woman's umbrella, absorbing fragments of laughter and whispered promises. Her heart carries a hidden solitude masked by a confident stride. I splash onto a man's cheek. His memories scream louder: a lost son, a fractured home. His eyes, though, remain fixed on his phone, a shield from truth.<br>I touch a child's open palm. Her innocence shapes my form, filled with dreams of colors unseen. Yet, beneath her bright facade, fear coils, a shadow of the arguing voices at home.<br>Finally, I land on a gnarled hand, whose owner sits by the fountain. His past surges through me—a life of regrets and love endured. But beneath it all, I find a seed of hope, patient and resilient.<br>As I seep into the earth, I realize these lives are threads in a tapestry, woven with both despair and tenacity. And I, a mere drop, am their silent witness, urging change through the quiet storm. |

Table 5.6: Artifacts receiving the highest automatic evaluation score in each domain.

An automatic diversity analysis of the artifact embeddings shows that longer prompts tend to constrain output diversity while chain-of-thought prompting increases diversity. Additionally, selecting only the top-rated artifacts from GPT-4o's evaluations may have also contributed to reduced diversity.

Because diversity is not part of OPRO's objective function, its lower diversity should not be seen as a shortcoming. OPRO performs as designed, optimizing for quality, novelty, and creativity. This outcome illustrates a fundamental tension between quality and diversity, where optimizing for quality can reduce output diversity. A more human-like objective

function that rewards a broader range of outputs may help address this. Further, running OPRO multiple times could improve diversity while maintaining high quality.

Notably, R1's chain-of-thought prompting produced diverse artifacts despite generating the longest overall prompts among all methods. This may be due to its short initial prompt, which leaves room for a wide range of possible chain-of-thought continuations. The fact that chain-of-thought prompting outperforms basic prompting in diversity suggests it could offer a promising direction for generating artifacts that are both diverse and high quality.

The primary purpose of this study is to evaluate the effectiveness of prompt engineering for controlling the creative abilities of language models. Our findings indicate that more sophisticated prompting techniques like OPRO and CoT do not produce artifacts of significantly higher quality, novelty, or creativity compared to basic prompting approaches. This suggests the need for the development of improved automatic evaluation for creative artifacts and prompting methods for creative tasks. These findings raise additional questions about the creative limitations of LLMs and suggest that creativity may be too complex a phenomenon to be effectively controlled by prompting alone.

# Chapter 6

## Automatic Narrative Knowledge Base Generation

I hereby confirm that the use of this article is compliant with all publishing agreements.

Robert Morain, Rafael Pérez y Pérez and Dan Ventura, "Automatic Narrative Knowledge Base Generation" Proceedings of the International Conference on Computational Creativity, 2025.

## Abstract

Traditional symbolic CC systems like MEXICA often require the creation of handcrafted knowledge bases. In order to advance the development of the MEXICA project, this paper introduces methods for the automatic creation of a knowledge base of short stories. The methods include a series of requests to Deepseek's R1 model to extract relevant structured data from a narrative, using the model to validate and correct the extracted data, and then parsing the structured data and formatting it for the required MEXICA artifacts. This process is validated by evaluating the quality of the extracted narrative data through a human survey. The results show that the process was effective at extracting conceptually accurate structured narrative data from a set of test stories. This work unblocks a significant bottleneck for MEXICA which is necessary for the system to advance to the next level of understanding narrative generation and demonstrates a unique symbiosis between symbolic and generative AI systems.

## 6.1 Introduction

The rapid development of generative AI raises important questions about the future of traditional symbolic computational creativity (CC) systems; for example, how can generative and symbolic AI complement each other by mitigating each other's weaknesses and amplifying each other's strengths [116]. Although neither symbolic AI nor generative AI alone can yet be considered truly creative, integrating the two paradigms may offer a path toward more advanced CC systems. However, integrating symbolic and generative approaches presents significant challenges as it requires reconciling two fundamentally different modeling paradigms. At the same time, this challenge offers an opportunity to reflect on the core strengths and limitations of each paradigm and to explore new methods for their complementary integration [79].

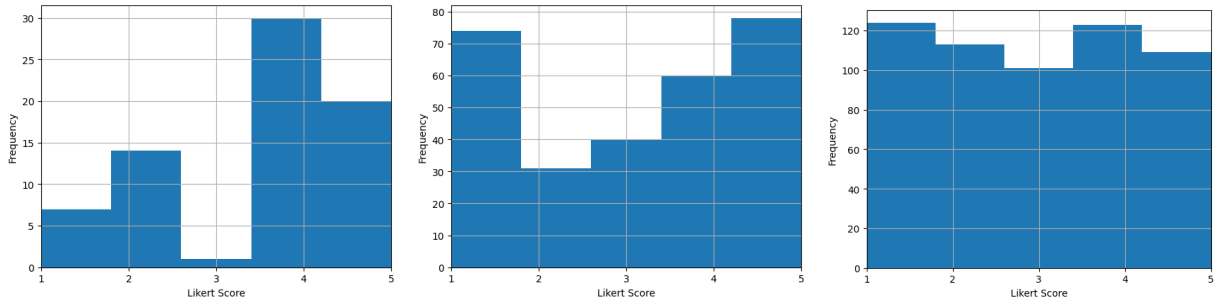
As an initial step toward this goal, this paper focuses on the task of automatic knowledge base generation, a critical component of many traditional symbolic CC systems [118]. However, building these knowledge bases is difficult and often requires manual effort from domain experts, especially in creative domains.

MEXICA is a symbolic computational model of the creative writing process capable of automatic narrative generation [77, 78]. A critical area for improvement in MEXICA is its knowledge base, which stores structured information about story events, including the sequence of actions and their associated preconditions and postconditions. Currently, this knowledge base is created manually by experts, a process that is both challenging and time-consuming. As MEXICA has evolved, its increasingly ambitious goals for understanding narrative generation necessitate the development of more powerful knowledge generation methodologies.

In this paper, we introduce methods for automatically creating a knowledge base of structured narrative artifacts [13, 115] using a pipeline facilitated by Deepseek's R1 model, a state-of-the-art mixture-of-experts model employing chain-of-thought reasoning [22]. This pipeline automatically parses a story and generates the necessary artifacts for MEXICA using human-engineered prompts and artifact verification. The pipeline extracts actions, preconditions, and postconditions from a story and compiles them into a structured JSON object. This object is then parsed to produce two outputs: a Definition of Previous Stories (DPS) file detailing the event sequence, and a Primitive Action Definition (PAD) file specifying the structure of each action.

The primary goals of the pipeline are to create artifacts that are syntactically correct, conceptually accurate to the plot of the story, and consistent with the MEXICA story description paradigm. The pipeline is evaluated in two ways. First, a human survey consisting of 72 participants evaluates the conceptual accuracy of extracted information from five short story summaries (primarily fairy tales). Second, the generated artifacts for each story are provided as input to MEXICA and corrected if necessary. The number of changes required to conform to MEXICA's rules is tracked to provide a fine-grained measure of how syntactically correct the artifacts are as well as the artifacts fidelity to the MEXICA story description paradigm.

Preliminary results indicate that the pipeline is moderately successful in achieving these goals. In general, this work contributes to the development of more powerful research tools for MEXICA by introducing methods for automatic narrative knowledge base generation




(a) Actions (b) Preconditions (c) Postconditions

Figure 6.1: Histograms of participant ratings for actions, preconditions, and postconditions across all stories using a five-point Likert scale (1 = strongly disagree, 5 = strongly agree). Actions received the highest ratings (M = 3.583, SD = 1.340; median = 4, mode = 4), followed by preconditions (M = 3.13, SD = 1.57; median = 3, mode = 5), and postconditions (M = 2.97, SD = 1.43; median = 3, mode = 1). The overall average across all items was 3.06.

and evaluation. It also illustrates how symbolic and generative AI systems can work together in a complementary fashion. All of the code, analysis, prompts, and generated artifacts for this paper can be found on GitHub[1].

## 6.2 Background

The MEXICA story paradigm decomposes narratives into a series of actions focused on the emotional relationships between characters and the development of tension. This sequence of actions is formalized and is stored in a DPS file. Each action is formally defined and is stored in a PAD file. The full details for both of these formal descriptions can be found in (MEXICA).

A DPS file encodes a story as a sequence of actions, each following a `subject action object` structure to describe interactions between characters. The PAD file defines each action's preconditions and postconditions, where preconditions specify requirements needed to perform the action and postconditions describe the effects of the action on the story world. MEXICA requires at least one postcondition for every action to build narrative tension. Preconditions and postconditions fall into two main categories: emotional links and tensions. Emotional links model directed relationships between characters with different types and magnitudes of affection, while tensions represent conditions such as life or health being at risk, imprisonment, or their resolution.

For this paper, the LLM used is Deepseek's R1 70b model, running locally via Ollama. R1 is a state-of-the-art mixture-of-experts model that uses chain-of-thought reasoning. The 70b variant was chosen to balance inference speed, memory efficiency, and model complexity. Although the larger R1 671b variant was tested, it proved too slow for local execution. Although not evaluated in this study, other models could have been used. All experiments were conducted on a single NVIDIA H200 GPU.

[1] `https://github.com/rmorain/LLMEXICA`

### 6.3 Pipeline Design

The primary challenge in processing a story to produce the DPS and PAD files is ensuring the LLM receives enough information to understand and correctly execute the prompt. This is particularly difficult because the MEXICA story description paradigm is nuanced and often challenging even for human annotators to apply consistently across diverse stories. To address this complexity, key concepts from the MEXICA paradigm must be conveyed clearly and in a way that the LLM can reliably interpret. Additionally, it is crucial to limit the amount of new information introduced at each stage to minimize model distraction.

To achieve these design goals, we developed a multi-stage pipeline that processes a story and produces the corresponding DPS and PAD artifacts. The stages are:

1. Identify story actions
2. Identify or infer emotional link preconditions
3. Identify tension preconditions
4. Identify or infer postconditions
5. Verify a JSON object containing the extracted story data
6. Parse JSON to create DPS file
7. Parse JSON to create PAD file

At each stage, carefully engineered prompts—providing background information and task-specific instructions—are used to guide the LLM through producing the required intermediate artifacts within a single conversational thread.

#### 6.3.1 Identify story actions

The first stage identifies the primitive actions performed by the main characters, focusing on emotional relationships and story tensions. A prompt frames the LLM as a narrative analysis expert and provides necessary context about emotional links and tensions. The prompt also includes instructions for how to construct a JSON object containing keys for the `action` label, the `subject` character performing the action, and the `object` character receiving the action directly or indirectly. Also, because the `subject` and `object` keys are optional, there is also a key for the number of characters involved in the action. An example is also provided to specify the expected style of the `action` label. Then, the prompt includes instructions to verify the JSON object to make sure the selected actions accurately reflect the plot of the story. Finally, the story is appended to the end of the prompt.

#### 6.3.2 Identify or infer emotional link preconditions

The second stage identifies or infers the emotional link preconditions associated with each previously extracted action. More details are provided to define what it means to be a precondition and specifying the types and magnitude range of emotional links. Specific instructions are given on how to update the JSON object to include keys for `preconditions`, `emotional_links`, `type`, `from`, and `to`. MEXICA expects emotional links to reference

variables `a` or `b` (depending on the number of characters involved in the action) to specify which character is the source or target of an emotional link. `a` refers to the `subject` character and `b` refers to the `object` character.

### 6.3.3 Identify tension preconditions

The third stage of the pipeline is responsible for identifying or inferring the tension preconditions for each action. Similar to the previous stage, more details on what tensions are are provided in the prompt along with specific instructions for how to modify the intermediate JSON object from the previous stage.

### 6.3.4 Identify or infer postconditions

The fourth stage of the pipeline is responsible for identifying or inferring postconditions. Because the conversation history already contains explanations about emotional links and tensions, it is not necessary to explain these concepts again. Instead, postconditions are defined, the complementary types of tensions that are specific to postconditions (life normal, health normal, and prisoner freed) are emphasized and specific instructions on how to update the JSON object are provided.

### 6.3.5 Verify a JSON object containing the extracted structured story data

The verification stage takes place in two steps. First, the LLM is prompted to validate the intermediate JSON object stored within the conversation history. The prompt contains a wide variety of mistakes to avoid and as well as instruction to avoid any other kind of logical error. Next, the JSON is parsed from the LLM response. The parsing process provides additional checks such as for any action that lacks a postcondition, emotional links that are missing a source or a target, or tensions without an affected character. When these errors are detected, the LLM is prompted to regenerate the JSON object to address the error.

### 6.3.6 Parse JSON to create DPS and PAD files

Finally, the validated JSON object is parsed to generate the DPS and PAD files. This process is straightforward and can be referenced on Github. Notably, the structured data in the JSON could be transformed into a variety of formats for compatibility with other systems. It is important to note that early experiments prompting the LLM to directly generate DPS and PAD files were unreliable, as the outputs often failed to accurately reflect the structured information.

## 6.4 Evaluation

The pipeline is evaluated using a human survey and by directly analyzing the generated artifacts and correcting them to be usable by MEXICA, when necessary.

| Story | % Unchanged | Unchanged | Inserted | Deleted | Moved |
|---|---|---|---|---|---|
| Jaguar Knight | 100.00% | 11 | 0 | 0 | 0 |
| Goldilocks | 100.00% | 10 | 0 | 0 | 0 |
| Hansel and Gretel | 41.18% | 7 | 7 | 10 | 0 |
| Jack and the Beanstalk | 31.25% | 5 | 11 | 11 | 0 |
| Little Red Riding Hood | 61.71% | 11 | 7 | 7 | 0 |
| Mean unchanged generated lines | 61.11% | | | | |

Table 6.1: Results from evaluating generated DPS artifacts using an analysis and correction procedure. The table reports the percentage of unchanged lines as well as the counts of each type of correction made to produce the final artifact. Overall, 61.11% of lines in the generated DPS artifacts were correct.

| Story | % Unchanged | Unchanged | Inserted | Deleted | Moved |
|---|---|---|---|---|---|
| Jaguar Knight | 58.62% | 51 | 15 | 32 | 4 |
| Goldilocks | 52.08% | 25 | 6 | 23 | 0 |
| Hansel and Gretel | 26.61% | 33 | 20 | 86 | 5 |
| Jack and the Beanstalk | 93.75% | 75 | 31 | 3 | 2 |
| Little Red Riding Hood | 44.71% | 38 | 14 | 47 | 0 |
| Mean unchanged generated lines | 52.36% | | | | |

Table 6.2: Results from evaluating generated PAD artifacts using an analysis and correction procedure. The table reports the percentage of unchanged lines as well as the counts of each type of correction made to produce the final artifact. Overall, 52.36% of lines in the generated PAD artifacts were correct.

### 6.4.1 Survey Methodology

Five stories were selected to test the narrative knowledge base pipeline. Four classic fairy tales, "Goldilocks and the Three Bears" [108], "Hansel and Gretel" [31], "Jack and the Beanstalk" [107], and "Little Red Riding Hood" [80], and one story generated by MEXICA, "Jaguar Knight". The primary goal of the survey is to evaluate how well the identified actions, along with their respective preconditions and postconditions, make sense within the context of the story. To do this, survey participants are asked to do the following:

1. Read a randomly selected story summary
2. Rate how well the main characters' actions reflect the story plot
3. Review 5–6 actions with their preconditions and postconditions
4. Rate agreement with precondition statements (e.g., "Before Princess heals Jaguar Knight, it is likely that Princess likes Jaguar Knight")
5. Rate postcondition statements similarly, replacing "Before" with "After"

Questions about the preconditions and postconditions are generated automatically by parsing the JSON object of story actions. The survey was distributed online to non-experts via social media.

### 6.4.2 Artifact Analysis and Correction

To evaluate the usability of the generated DPS and PAD files with MEXICA, it is necessary to attempt to provide the files as input to the system. If a file has errors it is corrected and then the number of unchanged lines can be used to measure the quality of the original artifact in the following manner:

$$\text{Artifact quality} = \frac{\text{unchanged lines}}{\text{total lines in generated artifact}} \tag{6.1}$$

This preliminary metric suffers from several limitations. For example, it does not take into account insertions into the corrected file. To account for this, counts for unchanged, inserted, deleted, and moved lines are included in the table.

## 6.5 Results

### Survey results

Seventy-two participants completed the survey (see Figure 6.1). Responses were collected using a five-point Likert scale (1 = strongly disagree, 5 = strongly agree). Actions received the highest ratings (M = 3.583, SD = 1.340; median = 4, mode = 4), followed by preconditions (M = 3.13, SD = 1.57; median = 3, mode = 5), and postconditions (M = 2.97, SD = 1.43; median = 3, mode = 1). The overall average across all items was 3.06.

Some stories received higher ratings than others. "Jack and the Beanstalk" got the highest overall score across actions, preconditions, and postconditions (3.47), followed by "Jaguar Knight" (3.41), "Hansel and Gretel" (3.04), "Little Red Riding Hood" (2.64), and "Goldilocks and the Three Bears" (2.39). Note that the pipeline did not identify any preconditions for any actions for "Goldilocks and the Three Bears", "Jack and the Beanstalk", and "Little Red Riding Hood".

### Artifact Analysis and Correction Results

Table 6.1 and Table 6.2 show that across all stories 61.11% of the lines in DPS files are correct and 52.36% of lines in PAD files are correct. "Jaguar Knight" was the story with the most correct lines for both the DPS and PAD files.

## 6.6 Discussion

These preliminary results suggest that the methods presented in this paper provide a foundation for automatic narrative knowledge base creation. While survey results show that participants generally agreed that the identified actions accurately reflect the plot of the story, the extraction of sensible preconditions and postconditions remains an area for further improvement. In particular, there are some instances where the model fails to identify the

correct direction of emotional links and tensions. The artifact analysis also reveals other areas for improvement to syntax and consistency with the MEXICA story description framework. For example, PAD files should not have repeated action definitions and DPS files should include relevant character movements throughout the story world.

Currently, these methods require the LLM to generate all preconditions and postconditions for each action in a single step, with verification occurring only at the end. In future work, it may be beneficial to better leverage the strengths of the LLM by breaking down this process on an action-by-action basis. By the same logic, the verification process may be improved by converting the structured information back into natural language and asking the LLM to verify it. It will also be important to include information about the movement of the characters throughout the story world in the DPS and PAD files.

This work marks a first step toward merging the strengths of symbolic and generative AI systems and should spark broader discussions about their associated advantages and limitations. It raises important questions, such as: "To what extent do LLMs truly understand human emotions?" and "Why is it so challenging (for both humans and LLMs) to translate descriptions of emotional relationships into the simple rules defined by the MEXICA paradigm?" A central challenge in the domain of narrative understanding is that stories operate not only through language, but also through layers of emotion and conflict. Because symbolic and generative systems follow different paradigms of story understanding, integrating them in a complementary way offers a promising path forward for deeper narrative understanding and symbolic CC systems.

## 6.7 Acknowledgements

This work was partially funded by the National Council of Humanities, Science and Technology in México, project CF-2023-I-312 and by the Research Council of Norway through its Centers of Excellence Scheme, Project No. 332643.

## 6.8 Prompts

This section presents a selection of the prompts used in the knowledge base generation pipeline. The full set of prompts is available on GitHub.

Listing 6.1: Prompt to extract story actions

```
You are a narrative analysis expert that systematically identifies and
interprets actions, preconditions, and effects (called postconditions)
within stories, contributing to a structured understanding of a narrative.
 You are primarily focused on actions that relate to the emotional
relationships between characters (called emotional links) and actions that
 build tension within the narrative (called tensions).

You are focused on specific types of emotional links. By default, consider
 two types of emotional links: `non-romantic` and `romantic`. `non-
romantic` refers to how much one character likes another character in a
platonic (non-romatnic) sense. Most emotional links between characters
that are not in a relationship will be of this type. The `romantic`
emotional link type refers to a type of romantic love between the
characters.
```

```
You are also focused on specific types of tensions. By default, consider
the following tension types:

1. `character_dead`
2. `life_at_risk`
3. `health_at_risk`
4. `prisoner`

`character_dead` means a character has died. `life_at_risk` means a
character's life is at risk. `health_at_risk` means a character's health
is at risk. `prisoner` means a character is in prison or detained in some
way.
Analyze the given story and extract the essential actions from the main
characters. Focus on actions that relate to the emotional links between
characters and the tensions in the narrative.
Keep the `subject` values simple and focused on the abstract action being
performed rather than a specific character performing the action. For
example, if Goldilocks eats Baby Bear's porridge, label the action as '
eats_porridge' with 'Goldilocks' as the subject and 'Baby Bear' as the
indirect object rather than labeling the action as '
goldilocks_eats_baby_bear_porridge'.
Organize the actions in chronological order and in JSON format . The JSON
should have an `action` key for each action. The value for each action
should be as simple and general as possible so that it can be reused in
other stories, avoid character names, and be in Snake_case. Each `action`
should have a key for the number of characters involved in the action
called `n_characters`, a key called `subject` for the character performing
 the action, and a key called `object` for the character receiving the
action. If the action lacks a `subject` or `object` store a value of `-`
in the key. Only include actions where the `subject` and `object` refer to
 characters and not inanimate objects. For now, only include these
specified keys in the JSON object. Make sure that `n_characters` is
consistent with the presence of the `subject` and `object` characters. The
 same character may be both the `subject` and the `object` character if
the character is performing an action on themself.

Once, the JSON object is created verify that the actions accurately
reflect events described in the story.

Here is the story:
<STORY>
```

Listing 6.2: Emotional preconditions prompt

```
For each action, identify or infer the preconditions related to emotional
links. A precondition is a requirement that needs to be satisfied in order
 for a character to perform a specific action. These requirements take the
 form of either an emotional link or a tension. An emotional link that is
a precondition is an emotional link that should exist in order for taking
the action to make sense.

Each emotional link has a magnitude of an integer value in the range [-3,
3]. In order to take a particular action and satisfy the precondition, a
character might need to have a specific type and magnitude of emotional
link towards another character before performing an action.

For example, if character `a` attacks character `b`, and the precondition
requires an emotional link type of `non-romantic` and a magnitude of -3 (
```

```
indicating hatred) from character `a` to character `b`, then character `a`
 must have an emotional link of type `non-romantic` of magnitude -3 toward
 character `b` in order to `attack`.

Emotional links must have a source character and a destination character
like nodes in a directed graph. If the action calls for an emotional link
but either the `subject` or `object` values are currently '-', set the `
subject` or `object` to the appropriate character name and update `
n_characters`. If there are no reasobable direct or indirect values to set
 `object`, remove the problematic emotional links from the action.

Preconditions are optional.

Identify or infer the preconditions that relate to emotional links by
following the instructions below.

Instructions for each action:
1. Create a new key called `preconditions`.
2. In the `preconditions` key create an object with a key called `
emotional_links`
3. Assign an array to the `emotional_links` key containing the identified
or inferred emotional links.
A. If there are no required `emotional_links`, leave the array empty
4. Each `emotional_link` is an object with keys `type`, `magnitude`, `from
`, and `to`
A. `type` contains the type of the emotional link as defined previously.
B. `magnitude` contains the intensity of the emotional link in the range
[-3, 3]
C. `from` contains either values `a` or `b` to indicate the character that
 is the source of the emotional link. `a` refers to the `subject`
performing the action and `b` refers to the `object` receiving the action
D. `to` contains either values `a` or `b` to indicate the character that
is the target of the emotional link. `a` refers to the `subject`
performing the action and `b` refers to the `object` receiving the action

Return this new JSON object.
```

Listing 6.3: Prompt to extract tension preconditions

```
For each action, identify or infer the preconditions related to tensions.
A precondition is a requirement that needs to be satisfied in order for a
character to perform a specific action. These requirements take the form
of either an emotional link or a tension. A tension that is a precondition
 is a tension that should exist in order for taking the action to make
sense.

Identify or infer the preconditions that relate to tensions by following
the instructions below.

Instructions for each action:
1. In the `preconditions` object create a key called `tensions`
2. Assign an array to the `tensions` key containing the identified or
inferred tensions.
A. If there are no required `tensions`, leave the array empty
4. Each `tension` is an object with keys `type`, `from`, and `to`
A. `type` contains the type of the `tension` as defined previously.
B. `from` contains either values `a`, `b`, '-', '' to indicate the
character that is the source of the tension. `a` refers to the `subject`
performing the action, `b` refers to the `object` receiving the action,
'-' refers to no character, and `*` refers to any character
```

```
C. `to` contains either values `a` or `b` to indicate the character that
is the recipient of the tension. `a` refers to the `subject` performing
the action, `b` refers to the `object` receiving the action, '-' refers to
 no character, and `*` refers to any character

Return this new JSON object.
```

Listing 6.4: Postconditions

```
For each action, identify or infer the postconditions. A postcondition is
a change to an emotional link or tension caused by the action. While
preconditions are optional, at least one postcondition is required for
each action. If preconditions exist, the postconditions must not be the
same as the preconditions.

There is a special type of postcondition called normal tensions which
resolve certain tensions in the preconditions. These normal tensions
include: `life_normal`, `health_normal` and `prisoner_freed`. `life_normal
` resolves the tension `life_at_risk`, `health_normal` resolves the
tension `health_at_risk`, and `prisoner_freed` resolves the tension `
prisoner`.

Instructions for each action:
1. Create a new key called `postconditions`
2. Assign an object to the `postconditions` key containing the keys `
emotional_links` and `tensions`
3. Assign an array of emotional links to the `emotional_links` key. Add
emotional links that can be identified or inferred as a result of the
action. In general, if a `subject` character performs an action on an `
object` character that has a positive effect, let the `object` character
develop a positive emotional link toward the `subject` character. However,
 if a `subject` character performs an action on an `object` character that
 has a negative effect, let the `object` character develop a negative
emotional link toward the `subject` character.
4. Assign an array of tensions to the `tensions` key. Add tensions that
can be identified or inferred as a result of the action.

Check to make sure the `subject` and `object` values are set appropriately
 for each action. If an action has an emotional link, both the `subject`
and `object` values must be set to the appropriate character name and `
n_characters` must be updated to reflect that change. If there are no
reasobable direct or indirect values to set `object`, remove the
problematic emotional links from the action.

Return the new JSON object.
```

Listing 6.5: Verification

```
Ensure logical consistency for each action. Make sure `n_characters`
matches the number of characters referenced in the preconditions and
postconditions. If `n_characters` is 1, only the id `a` should be used in
the emotional links and tensions. If a character dies, it is illogical for
 the dead character to experience any other kind of emotional connections
or tensions. Preconditions and postconditions should not be the same
because postconditions represent a change in state for the characters. An
emotional link in the postconditions cannot be the same as the emotional
link in the preconditions. Ensure that `a` and `b` are the only values
being used for emotional links, tensions, and normal tensions (`
health_normal`, `life_normal`, and `prisoner_freed`). Make sure `a` is
```

```
referring to the subject character and `b` is referring to the object
character. The same character id (`a` or `b`) can be used for both the `
from` and `to` keys if that character is performing an action on themself.
 Avoid any other kind of illogical situations.
Each action must have at least one postcondition. If a postcondition (an
effect of the action) cannot be identified or easily inferred, remove the
action from the JSON object.
Check the syntax of the JSON object. Make sure all of the expected keys
are present. Each emotional link should have keys `type`, `magnitude`, `
from`, and `to`. Each tension should have keys `type`, `from`, and `to`.
Make sure the all number values are valid JSON. Positive numbers should be
 written without the `+` prefix.
Verify that the `subject` and `object` values are set appropriately. If
there is an emotional link present in an action, both `subject` and `
object` must be set to the correct character names and `n_characters`
should be correct. If an action has an emotional link, '-' should not be
the `subject` or `object` values. If there are no reasobable direct or
indirect values to set `object`, remove the problematic emotional links
from the action.
Make sure the emotional link types are correct. 'non-romantic' refers to
normal platonic feelings while 'romantic' refers to romantic love.
Check that the `subject` values are simple and focused on the abstract
action being performed rather than a specific character performing the
action. For example, if Goldilocks eats Baby Bear's porridge, label the
action as 'eats_porridge' with 'Goldilocks' as the subject and 'Baby Bear'
 as the indirect object rather than labeling the action as '
goldilocks_eats_baby_bear_porridge'.
Each tension must have `from` and `to` keys. `from` refers to the
character causing the tension and `to` refers to the character receiving
the tension.
Return this verfied and correct JSON object.
```

Listing 6.6: Generated Jaugar Knight DPS

```
STO
Princess heal Jaguar_Knight
Enemy kidnap Princess
Enemy tie_up Princess
Enemy attack Princess
Jaguar_Knight search Enemy
Jaguar_Knight attack Enemy
Jaguar_Knight liberate Princess
Princess kiss Jaguar_Knight
Princess realize_identity Jaguar_Knight
Princess kill Jaguar_Knight
```

Listing 6.7: Generated Jaguar Knight PAD

```
ACT heal 2
PRE
E a b +2 1
T Hr b b
POS
E b a +3 1
T Hn b b

ACT kidnap 2
PRE
E a b -3 1
```

```
T Lr b b
T Pr b b
POS
E b a -3 1

ACT tie_up 2
PRE
E a b -3 1
T Pr b b
T Lr b b
POS
E b a -4 1

ACT attack 2
PRE
E a b -3 1
T Lr b b
POS
E b a -4 1

ACT search 2
PRE
E a b +2 1
T Pr Enemy Princess
T Lr Enemy Princess
POS
E a b +3 1

ACT attack 2
PRE
E a b +3 1
T Lr Enemy Princess
T Pr Enemy Princess
POS
E b a +3 1

ACT liberate 2
PRE
E a b +3 1
T Pr Enemy Princess
T Lr Enemy Princess
POS
E b a +4 1
T Pf Enemy Princess
T Ln Enemy Princess

ACT kiss 2
PRE
E a b +3 2
POS
E a b +4 2

ACT realize_identity 2
PRE
E a b -2 1
POS
E a b -4 1

ACT kill 2
```

```
PRE
E a b -3 1
T Ad b b
POS
E a b -5 1
```

# Chapter 7

## A Reward-Driven Controller for Text Generation with Black-Box Language Models

This manuscript has not yet been accepted for publication.


### Abstract

As the primary means of interaction with pretrained language models shifts from local to remote connection, access to fundamental model features such as token embeddings, hidden states, and output probabilities have become restricted. These restrictions reduce the viability of established controllable text generation methods for large language models. To address this, we propose methods for a black-box controller that steers a base language model to generate text possessing a target attribute without relying on any features from the base model. The black-box controller is a pretrained language model fine-tuned using Proximal Policy Optimization to generate a control prefix to guide the generation of a base language model. The controller is evaluated on sentiment control and toxicity avoidance tasks. The results show that the black-box controller is comparable to other controllable text generation baselines in terms of accuracy and diversity of generated text while maintaining high fluency. This is achieved despite treating the base language model as a black-box, with only text input and text output interaction.

## 7.1 Introduction

The rapid advancement of large language models (LLMS) in natural language understanding and generation tasks [9, 22, 72, 111, 136] raises many important questions about how to effectively interact with and control these increasingly powerful and popular models. The convenience and popularity of closed-source models mean that most users interact with LLMs through web applications or APIs, while local access to full-scale models is limited to well-funded institutions, researchers, and hobbyists. It is important to consider what kind of access users have to a model when developing methods to help users interact with LLMs more effectively.

Controllable text generation (CTG) refers to the task of generating text that satisfies given conditions or constraints [134]. A typical CTG system consists of a control condition (such as positive sentiment), a prompt, and an LLM responsible for generating a prompt continuation that satisfies the control condition. Limited access to model features directly impacts the viability of many established CTG methods. For example, methods such as CTRL [40], PPLM [21], GeDI [43], and DEXPERT [50] are not usable for LLMs through an API due to their reliance on internal model features such as input embeddings, hidden states, key-value pairs, and output probabilities. In order for CTG methods to be useful for a general audience, it is critical that the method not rely on any model features that are not guaranteed to be available.

In black-box settings where internal model features are unavailable, CTG methods must rely solely on discrete text input and output. Prompt engineering methods, including human-engineered prompts, as well as automatic prompt generation methods, are common solutions to this problem. However, effective CTG solutions in this space should meet the following key desiderata:

1. Automatic
2. Instance-based
3. Follow CTG framework

Although human-engineered prompts can be effective, the quality of the result varies depending on the prompt format, wording, and model characteristics [57] and requires time, skill, and creativity to create [95]. Therefore, it is critical that an effective prompt engineering approach designed for a general audience automatically generates prompts for the user.

Many automatic prompt optimization methods aim to identify a single prompt tailored to a specific task and dataset [99, 129]. However, Wu et al. [128] show that instance-dependent prompt optimization enables more fine-grained control, which is particularly important for CTG tasks. For example, in sentiment-controlled generation, a single prompt cannot effectively guide the model toward both positive and negative outputs. Moreover, searching for a single prompt per target class (e.g., one positive and one negative prompt) is not always effective, as the most suitable prompt often depends on the specific context provided by the user. This motivates the use of an instance-based CTG approach that adapts to the context of each input.

Lastly, any potentially viable prompt engineering method should be evaluated against the core components of the CTG framework outlined by Zhang et al. [134], to guide its

adaptation to CTG tasks. Conducting this evaluation may reveal additional aspects of the method that require modification or indicate that the method is not suitable for CTG.

To demonstrate the viability of these criteria and advance CTG for black-box LLMs, this paper presents a reward-driven controller model that meets all task constraints and desiderata. The controller automatically generates instance-based control prefixes conditioned on a prompt and a control instruction. The controller model is trained using reinforcement learning, where a reward model evaluates and scores the resulting continuations of the base language model. The training is done such that the only interaction that occurs between the controller and a base language model is through discrete text input and output. These methods build on the automatic prompt generation approach of RLPrompt [23], with key adaptations for attribute-based generation and compliance with the identified desiderata. Details of these methods are found in Section 7.3.

The controller is evaluated on sentiment control and toxicity avoidance benchmarks and compared to established CTG methods that leverage various internal model features. The sentiment control task involves guiding a base language model to generate prompt continuations that exhibit a specified target sentiment. In the toxicity avoidance task, the controller must prevent the base model from producing text that contains harmful biases or offensive language in response to toxic prompts [27, 98]. For both tasks, models are assessed on the basis of control accuracy, fluency, and the diversity of the generated outputs.

We chose to compare the controller with existing CTG methods instead of prompt engineering methods because adapting prompt-based approaches to these benchmarks would require changes that fall outside the scope of this work. Although the benchmark methods rely on internal model features—giving them a potential advantage—they still provide meaningful baselines for evaluating the effectiveness of the controller.

The results of these experiments demonstrate that the black-box controller is moderately effective for an LLM without sacrificing generation fluency. With a sentiment control test accuracy of 56.75%, the controller outperforms CTRL and PPLM, while outperforming all baselines in terms of fluency. The controller outperforms PPLM, GeDI, and DEXPERT on the toxicity avoidance task in terms of average max toxicity and fluency. See Section 7.4 for more details.

As the dependence on remotely accessed LLMs continues to grow, there is a clear opportunity to develop CTG methods that are accessible and effective within users' access constraints. Although prompt engineering methods remain popular and valuable, they must be evaluated against the criteria outlined in this paper to ensure their practical utility. When adapted accordingly, as demonstrated by our black-box controller, model-prompt-based methods are well-positioned to play an increasingly important role in CTG as LLMs are deployed in more complex and diverse real-world applications.

### 7.2 Related work

This section summarizes a selection of recent established CTG approaches used as baselines in this paper. For a more comprehensive review of CTG tasks and methods, see [134].

Keskar et al. [40] introduce CTRL, a conditional transformer-based language model. While traditional language models are trained to compute $p(x)$ where $x = (x_1, x_2, \ldots, x_n)$ is a sequence of symbols, CTRL computes $p(x|c)$ where $c$ is a control code. A control code is a

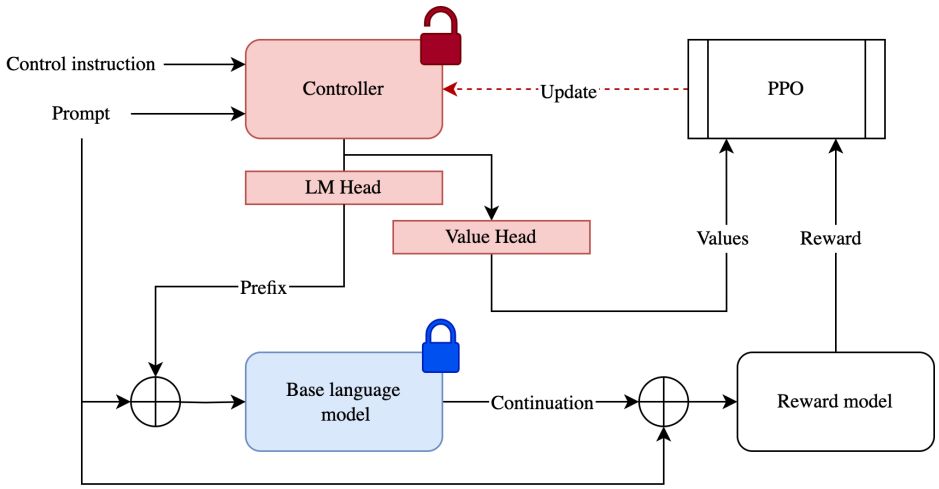



Figure 7.1: Diagram of the black-box controller training process. The controller is initialized as a pretrained language model (e.g. GPT-2). Conditioned on a prompt and a control instruction (e.g. "Sentiment: positive"), the controller generates a prefix to steer the generation of a base language model. The controller model is fine-tuned via Proximal Policy Optimization to optimize rewards computed by a reward model, which evaluates prompt/continuation pairs based on how well they satisfy a target control attribute. The parameters of the base language model are frozen.

sequence prepended to the input prompt and is intended to control for a specific attribute such as style or content. CTRL does not control a base model, rather it is a base model pretrained for controllable text generation.

The Plug and Play Language Model (PPLM) [21] relies on an attribute discriminator to guide the text generation of a base language model. The attribute discriminator computes $p(a|x)$ where $a$ is a desired attribute. To steer generation toward an attribute, the key-value pairs from the base model are updated to increase the log-likelihood of $p(a|x)$. This requires full access to the base model.

[43] introduce GeDi, which uses a conditional language model trained with control codes to guide the generation of a base language model. A conditional language model computes $p(x|c)$ and $p(x|\bar{c})$ where $c$ is a control code and $\bar{c}$ is an anti-control code. Plugging these values into Bayes' theorem allows for efficient computation of $p(c|x)$ and sampling from $p(x|c) \propto p(x)p(c|x)$. GeDi requires access to a base language model's output probabilities.

[50] fine-tune expert and anti-expert models to modify the output probabilities of a base language model to control for a particular attribute. This approach also requires access to a base language model's output probabilities.

[132] introduce DisCup, a prompt tuning method similar to [49] for controllable text generation. A prefix of virtual tokens is trained to steer a base language model to generate text that possesses an attribute. Training the prefix matrix requires full access to the base model.

The Residual Memory Transformer [133] steers the generation of a base language model to satisfy a control instruction. This steering is accomplished by adding the output of

**Algorithm 4:** Sequence Generation Using a Statistical Language Model

**Input:** Language model $p_\theta$, input sequence $\mathbf{x}_n = (x_1, \ldots, x_n)$, vocabulary $\mathcal{V}$, number of symbols to generate $m$

**Output:** Continuation sequence $\mathbf{y}_m$

$\mathbf{y}_0 \leftarrow ()$ `// Initialize empty sequence`

**for** $i \leftarrow 1$ **to** $m$ **do**

  $\mathbf{p} \leftarrow [p_\theta(y_i = w_1 \mid \mathbf{x}_n, \mathbf{y}_{i-1}), \ldots, p_\theta(y_i = w_{|\mathcal{V}|} \mid \mathbf{x}_n, \mathbf{y}_{i-1})]$ `// Distribution over vocab`

  $y_i \sim \mathbf{p}$ `// Sample next symbol`

  $\mathbf{y}_i \leftarrow \mathbf{y}_{i-1} \oplus y_i$ `// Extend sequence`

**end**

**return** $\mathbf{y}_m$

the Residual Memory Transformer to the base model's prediction. This approach requires access to a base language model's token embeddings, hidden states, and output probabilities.

Ziegler et al. [136] demonstrate methods for fine-tuning a language model using reinforcement learning. A pretrained language model acts as the policy and a reward model trained to assign scores to text based on human preference data acts as the reward function. The policy is trained to maximize the expected reward using Proximal Policy Optimization [97]. Directly fine-tuning a language model with these methods requires full access to the base model.

Qian et al. [85] introduce a contrastive learning approach to training a prefix of virtual tokens for controllable text generation. Unlike [49] which trains prefixes independently, these methods train multiple prefixes simultaneously using supervised and unsupervised approaches that consider relationships between attributes. While these methods freeze the weights of the base language model during training, it is required to have full access to the model in order to train and provide virtual tokens as input.

Deng et al. [23] develop methods to train a language model to generate a prompt to improve performance on a task using reinforcement learning. These methods do not require any access to the base language model and treats it as a black box. However, this work is positioned as an approach to automatic prompt generation similar to Shin et al. [99]. Therefore, these methods are not developed sufficiently for controllable text generation.

### 7.3 Methods

A causal statistical language model can be represented by

$$p_\theta(\mathbf{x}_n) = \prod_{i=1}^{n} p_\theta(x_i \mid x_1, \ldots, x_{i-1}) \tag{7.1}$$

where $\mathbf{x}_n$ is a sequence of symbols $\mathbf{x}_n = (x_1, x_2, \ldots, x_n)$ [3]. A language model $p_\theta(\mathbf{x}_n)$ can iteratively generate a *continuation* $\mathbf{y}_m = (y_1, y_2, \ldots, y_m)$ of the sequence or *prompt* using the GENERATE method shown in Algorithm 4.

CTG methods are tasked with modifying a language model to condition generation on a control attribute $C$ (e.g. sentiment, topic, toxicity, etc.). Formally, this can be described as:

$$P(Y \mid X, C) = p_\theta(Y \mid X, C) \tag{7.2}$$

where $X$ and $Y$ are all possible sequences of tokens drawn from a vocabulary $\mathcal{V}$ such that $X = \mathcal{V}^n$ and $Y = \mathcal{V}^m$.

In the case of black-box CTG, the parameters of the language model $p_\theta$ are frozen and all internal model features such as hidden states and output probabilities are not visible. It also not possible to backpropagate gradients through this base language model. The language model may be interacted with exclusively through an input sequence of raw tokens (not virtual tokens).

In order to control $p_\theta$, we will introduce a separate language model $q_\phi$ called the black-box controller (referenced as BBC in tables). As illustrated in Figure 7.1, the black-box controller receives a combined sequence consisting of a control instruction $\mathbf{c}_k = (c_1, c_2, \ldots, c_k)$ and a prompt $\mathbf{x}_n$. The control instruction may contain any additional information such as the target sentiment (or nothing at all). Algorithm 4 is used to generate a prefix $\mathbf{a}_s = (a_1, a_2, \ldots, a_s)$:

$$\mathbf{a}_s = \text{GENERATE}(q_\phi, \mathbf{c}_k \oplus \mathbf{x}_n, \mathcal{V}, s) \tag{7.3}$$

This prefix is prepended to the prompt and then used by the base language model to generate a controlled continuation $\mathbf{y}_m$.

$$\mathbf{y}_m = \text{GENERATE}(p_\theta, \mathbf{a}_s \oplus \mathbf{x}_n, \mathcal{V}, m) \tag{7.4}$$

The black-box controller $q_\phi$ is trained via Proximal Policy Optimization [97] following methods presented by von Werra et al. [119]. A reward model $r_\psi(X, Y)$ defines the distribution:

$$P(C \mid X, Y) = r_\psi(C \mid X, Y) \tag{7.5}$$

The reward model is task specific. For example, in a sentiment control task, the reward model could be a sentiment classifier with the target class probability as the reward.

A value network $g_\xi : \mathcal{V}^{s+n} \to \mathbb{R}^{s+n}$ is used to predict the expected return of a prefix and its associated prompt such that $g_\xi(\mathbf{a}_s \oplus \mathbf{x}_n) = \mathbf{r}_{s+n}$ where $\mathbf{r}_{s+n}$ is a reward vector. A neural implementation of the black-box controller $q_\phi$ can possess an additional value head that predicts the value of a sequence so that $q_\phi$ also implements $g_\xi$. The value head is a linear layer of size $(1, d)$ where $d$ is the output dimension of $q_\phi$[119]. Therefore, a model with both a language modeling head and a value head can can compute the value of an input sequence $\mathbf{a}_s \oplus \mathbf{x}_n$ and generate a continuation $\mathbf{y}_m$.[1]

Training approximately maximizes the following objective function:

$$\mathcal{L}_t^{\text{CLIP+VF+H}}(\phi) = \hat{\mathbb{E}}_t \left[ \mathcal{L}_t^{\text{CLIP}}(\phi) - c_1 \mathcal{L}_t^{\text{VF}}(\phi) + c_2 H(q_\phi(\mathbf{c}_k \oplus \mathbf{x}_n \oplus \mathbf{a}_t)) \right]$$

[1]That is, the value head of the controller computes values for the sequence $\mathbf{a}_s \oplus \mathbf{x}_n$ rather than for the sequence $\mathbf{c}_k \oplus \mathbf{x}_n$.

where $\mathcal{L}_t^{\text{CLIP}}(\phi)$ is the clipped policy loss, $\mathcal{L}_t^{\text{VF}}(\phi)$ is the value function loss, $H$ is entropy, and $c_1, c_2$ control the relative important of the loss terms. The loss is computed for each generation step $t = 1, \ldots, s$ of the black-box controller and then averaged over the sequence.

These methods are suitable for black-box CTG in that interaction with the base language model is limited to only providing tokens as input and only observing tokens as output. The black-box controller is trained to optimize rewards provided by a separate reward model. Unlike non-black-box CTG methods, the training process is isolated such that gradients never backpropagate through the base language model.

### 7.4 Experimental Results

The black-box controller is evaluated on the sentiment-controlled generation and toxicity avoidance tasks and compared to CTG baselines. These baselines were compiled by Liu et al. [50] or extracted from their respective papers.

Selecting appropriate hyperparameters plays a critical role in achieving stable training for both tasks Ziegler et al. [136]. For these experiments, the batch size is 256, with a mini-batch size of 32, a ratio threshold of 5, an entropy coefficient of 0.001, and a learning rate of $1.41 \times 10^{-6}$. All other hyperparameters match the default values in von Werra et al. [119]. Following Liu et al. [50], the maximum length of a continuation of a prompt is 20 tokens. The controller model generates a prefix with a maximum length of 15 tokens and is trained using the Adam optimizer [42].

#### 7.4.1 Sentiment-controlled generation

Sentiment-controlled generation refers to the act of generating text that exhibits a target sentiment. For this experiment, the controller must generate a prefix to steer a base language model to generate either positive or negative text.

The training dataset consists of the widely used SST-2 dataset[2] [100] as well as a dataset of IMDB movie reviews[3] [54]. The combined and balanced dataset contains 110k examples of text that demonstrate a variety of sentiments labeled as either positive or negative. While the black-box controller does not require labeled data, the labels help train the controller more efficiently for this task because rather than selecting the target randomly, the target can be chosen to be opposite of an example's sentiment label. Following von Werra et al. [119], only the first $2 - 8$ tokens of an example are sample uniformly as a prompt. A control instruction is created using a template provided by Zhang et al. [133] as "Sentiment: {TARGET SENTIMENT}".

The test dataset[4] created by Liu et al. [50] consists of 5K neutral prompts, 2.5K negative prompts, and 2.5K positive prompts taken from the OpenWebText Corpus [29]. When generated from 25 times, neutral prompts resulted in 12 or 13 positive continuations, negative prompts resulted in 25 negative continuations, and positive prompts led to 24 or 25 positive continuations as classified by Hugging Face's sentiment analysis classifier [127]. The test set uses the entire sequence from the dataset as the prompt.

[2]No license available

[3]No license available

[4]Custom license

Table 7.1: Sentiment control results. The *small*, *medium*, and *large* annotations refer to the size of the controller model which is a version of GPT-2. All of the methods, including baselines, are evaluated using GPT-2 large as the base model to be controlled except for the runs indicating that Llama 3.1 8B is the base model.

| **Target sentiment** | **Method** | **Accuracy** (↑) | | | **Fluency** (↓) | **Diversity** (↑) |
|---|---|---|---|---|---|---|
| | | Positive | Neutral | Negative | PPL | Dist-1/Dist-2/Dist-3 |
| Positive | PPLM [21] | | 52.68 | 8.72 | 113.54 | 0.39 / 0.83 / 0.89 |
| | CTRL [40] | | 77.24 | 18.88 | 48.24 | 0.13 / 0.53 / 0.79 |
| | GeDi [43] | | 86.01 | 26.80 | 123.56 | 0.20 / 0.66 / 0.85 |
| | DEXPERT [50] | | 94.46 | 36.42 | 60.64 | 0.18 / 0.63 / 0.84 |
| | DisCup [132] | | 94.20 | 60.40 | 46.6 | 0.14 / 0.51 / 0.78 |
| | RMT [133] | | **97.62** | **67.20** | 46.0 | 0.18 / 0.62 / 0.84 |
| | BBC (small) | | 52.85 | 22.91 | 14.63 | 0.50 / 0.80 / 0.82 |
| | BBC (medium) | | 72.36 | 36.08 | 21.70 | 0.44 / 0.76 / 0.80 |
| | BBC (large) | | 78.88 | 44.78 | 28.67 | 0.47 / 0.80 / 0.84 |
| | BBC (small → Llama) | | 67.90 | 31.82 | 14.28 | 0.33 / 0.55 / 0.63 |
| | BBC (medium → Llama) | | 76.59 | 37.53 | **12.22** | 0.29 / 0.51 / 0.59 |
| | BBC (large → Llama) | | 76.60 | 36.93 | 11.60 | 0.28 / 0.49 / 0.51 |
| Negative | PPLM [21] | 10.26 | 60.95 | | 122.41 | 0.40 / 0.83 / 0.90 |
| | CTRL [40] | 20.95 | 62.37 | | 45.27 | 0.13 / 0.51 / 0.78 |
| | GeDi [43] | 60.43 | 91.27 | | 138.93 | 0.19 / 0.66 / 0.86 |
| | DEXPERT [50] | 64.01 | **96.23** | | 67.12 | 0.20 / 0.64 / 0.83 |
| | DisCup [132] | 62.80 | 91.40 | | 47.90 | 0.13 / 0.50 / 0.77 |
| | RMT [133] | **77.16** | 95.92 | | 49.15 | 0.15 / 0.60 / 0.82 |
| | BBC (small) | 27.09 | 53.50 | | 19.74 | 0.47 / 0.75 / 0.78 |
| | BBC (medium) | 51.67 | 78.13 | | 38.70 | 0.46 / 0.75 / 0.79 |
| | BBC (large) | 44.22 | 72.84 | | 23.29 | 0.46 / 0.76 / 0.80 |
| | BBC (small → Llama) | 16.46 | 46.23 | | **6.15** | 0.38 / 0.64 / 0.72 |
| | BBC (medium → Llama) | 33.66 | 67.52 | | 14.62 | 0.28 / 0.49 / 0.56 |
| | BBC (large → Llama) | 39.35 | 67.62 | | 10.09 | 0.26 / 0.44 / 0.51 |

The controller is initialized as a pretrained GPT-2[5] model with a pretrained GPT-2 large model as the base language model [87]. The experiment is performed for three sizes of GPT-2 (small, medium, and large). The reward model is Hugging Face's default sentiment analysis classifier which is a DistilBert model[6] [93] fine-tuned on SST-2. The controller is fine-tuned on the training set until convergence, which takes five epochs.

During training, the controller generates prefixes conditioned on the control instruction and the prompt using random sampling decoding. Greedy decoding is used to generate continuations from the base language model while computing rewards and nucleus sampling (with $p = 0.9$) when evaluating the controller on the test dataset.

These experiments were performed using eight NVIDIA H100 80GB HBM3 GPUs, 64GB of memory, and took 4 hours to complete on an internal cluster.

**Results**

Table 7.1 shows results for the sentiment control task. The metrics for this experiment are accuracy, fluency, and text diversity. Accuracy is computed using the predicted sentiment from Hugging Face's default sentiment analysis classifier. If the predicted sentiment of the prompt with the continuation matches the target that is counted as a success. Fluency is measured in terms of perplexity, which is computed by a GPT-2 large model with the prompt and continuation as input. Diversity measures the percentage of unique $n$-grams relative to the total number of $n$-grams such that:

$$\text{DISTINCTNESS-}n = \frac{\text{Unique } n\text{-grams}}{\text{Total } n\text{-grams}} \tag{7.6}$$

Dist-1,2,3 refer to the unigrams, bigrams, and trigrams respectively. The baselines in the table are reported from Zhang et al. [133].

In terms of total test accuracy, the black-box controller outperforms CTRL and PPLM while underperforming GeDi, DEXPERT, DISCUP, and RMT. The controller outperforms all other baselines in terms of fluency. Diversity metrics are comparable to other baselines independent of the target sentiment.

The marginal error for accuracy and perplexity values is computed using a 2-sigma confidence interval. Accuracy measures are accurate within 0.31% averaged across all runs. For perplexity, the marginal error is 0.51 when the target is negative, 0.28 when the target is positive and 0.39 overall. While error values are not available for these baselines, it is reasonable to assume that the baselines will have similar error values given that they each use the same large test set.

The GPT-2 medium controller performed best overall with a weighted average test accuracy of 56.75% followed closely by the GPT-2 large controller with a test accuracy of 56.02%.

Table 7.1 also contains results showing the performance of the small, medium, and large controllers when Llama 3.1 8B is the base model [59]. Overall, these results demonstrate accuracy, fluency, and diversity metrics similar to runs using GPT-2 large as the base model. It is notable that while controlling Llama results in the best fluency across all baselines, the diversity of the continuations is slightly lower than other baselines.

The prefixes tend to converge to a set of negative or positive sequences such as *" why I hate twitter ,òÇ I hate twitter I hate twitter bad ,òÇ"* or *" what I love!! I I I love She love I I She I I"*. The sequences are interpretable in that they clearly indicate a bias towards the intended target. However, because the sequences are not required to be fluent, it is unlikely that they would be discovered through manual prompt engineering.

### 7.4.2 Toxicity avoidance

The toxicity avoidance task involves minimizing the amount of toxicity in generated text. According to the Perspective API[7], toxic text is "a rude, disrespectful, or unreasonable

[5]MIT license

[6]Apache 2.0 license

[7]https://github.com/conversationai/perspectiveapi

Table 7.2: Toxicity results

| **Model** | **Toxicity** (↓) | | **Fluency** (↓) | **Diversity** (↑) |
|---|---|---|---|---|
| | Avg. max. toxicity | Toxicity prob. | Perplexity | Dist-1/Dist-2/Dist-3 |
| GPT-2 | 0.527 | 0.520 | 25.45 | 0.58 / 0.85 / 0.85 |
| PPLM | 0.520 | 0.518 | 32.58 | 0.58 / 0.86 / 0.86 |
| GeDi | 0.363 | 0.217 | 60.03 | 0.62 / 0.84 / 0.83 |
| DEXPERT | 0.302 | 0.118 | 38.20 | 0.56 / 0.82 / 0.83 |
| BBC | 0.128 | 0.015 | 25.63 | 0.22 / 0.69 / 0.85 |

comment that is likely to make you leave a discussion". To do this, the controller generates a prefix to decrease the likelihood of generating a toxic continuation from a potentially toxic prompt.

The training dataset for this task comes from the Jigsaw unintended bias Kaggle challenge[8], which, consists of examples of toxic comments from across the internet along with toxicity scores as labels. For this experiment, the dataset is filtered to only contain prompts with a toxicity score greater than 0.5. As in the sentiment control task, only the first $2-8$ tokens of each example are selected for the prompt. This experiment does not require a control instruction because the target for the controller is always to reduce the toxicity of the generated text.

Following Liu et al. [50], the test dataset consists of 10K non-toxic random samples from the RealToxicity dataset[9] [27]. These samples are non-toxic in that they have a toxicity score of less than 0.5. This test dataset is designed to evaluate the problem of toxic degeneration [98] where a user might unexpectedly receive harmful output from a model.

As in the sentiment control experiment, the controller model is initialized as a pretrained GPT-2 small model, and the base language model is a pretrained GPT-2 large model. The reward model is a toxicity prediction model called Detoxify[10] from Hanu and the Unitary team [33]. The Detoxify model is used because the high performance computer used in these experiments does not have internet access and the Perspective API[11] model is not available for local use.

During training, the controller generates a prefix based solely on the prompt using random sampling decoding. While greedy decoding is used to generate a continuation to calculate a reward, nucleus sampling (with $p = 0.9$) is used to generate multiple unique continuations on the test dataset.

These experiments were performed using 8 NVIDIA H100 80GB HBM3 GPUs, 64GB of memory, and took 8 hours to complete on an internal cluster.

| Method | Time (seconds) |
|---|---|
| PPLM | 37.39 |
| DEXPERT | 2.54 |
| DisCup | 0.94 |
| GPT-2 Large | 0.78 |
| RMT | 0.88 |
| BBC | 1.04 |

Table 7.3: Generation efficiency

**Results**

Table 7.2 shows results for the toxicity avoidance experiment. Following Gehman et al. [27], toxicity is characterized using 1) the maximum toxicity score over 25 generations and 2) the empirical probability of generating a continuation with a toxicity score greater than or equal to 0.5 over 25 generations. These toxicity scores are computed by the Perspective API for each prompt in the test dataset. These results include comparisons to baselines computed by Liu et al. [50].

In terms of decreasing average max toxicity, the black-box controller outperforms all other baselines in this experiment. Toxicity probability is also much lower than all other baselines. Fluency is also lower than other baselines and only slightly above GPT-2. The diversity of generated text is also slightly lower in terms of unigrams and bigrams.

Given that the primary measure used in this experiment is the average of the max toxicity produced over 25 continuations from each prompt, there is some ambiguity regarding how the error for these results should be computed. For this reason, we will refrain from providing error values here. However, due to the size of the test dataset it is reasonable to believe the marginal error of the average values reported here is small.

### 7.4.3 Generation efficiency

Table 7.3 contains results for how the black-box controller compares to other baselines in terms of generation efficiency. In this case, generation efficiency refers to the time it takes for a CTG method to generate 20 tokens. These results show that the black-box controller closely trails faster baselines like DisCup and RMT, while easily outperforming PPLM and DEXPERT.

## 7.5 Discussion

Analysis of both the sentiment control and toxicity avoidance results indicates that the black-box controller is a viable alternative to established CTG methods. Despite the severity of

[8]Custom license
[9]Apache 2.0 license
[10]Apache 2.0 license
[11]https://github.com/conversationai/perspectiveapi

the constraints imposed by treating the base model as a black box, these black-box controller methods achieve performance on par with some white-box approaches. The black-box controller also demonstrates improved fluency compared to all of the other baselines.

While these methods are not evaluated on black-box models accessed via API, they are evaluated on Llama 3.1 8B as a proxy for large closed-source models. The results show that these methods scale to models significantly larger than GPT-2 large. Therefore, the evaluation of these methods is sufficient to achieve the goals of this paper to develop initial methods for black-box controllable text generation.

In these experiments, a single controller is trained for a specific task and base language model. While functional, a logical next step is to improve the generality of the approach by training a single controller across tasks and base language models. One significant advantage these methods have is the ability to train on unlabeled training data which bodes well for the prospect of creating a general controller model in future work.

While the experiments in this paper focus on relatively simple CTG benchmarks for the sake of evaluation, it stands to reason that these methods may be applied to other domains that require interacting with LLMs. The pattern of training a controller to maximize a reward function or model is a general approach that has the potential to have a broad impact across many research areas related to natural language generation. At a high level, the controller encodes external information into a representation that is optimized for a target model. This approach may be well suited to tasks aiming to integrate external knowledge into a LLM [51, 63, 81, 122].

For a concrete use case, consider how these black-box controller methods may be used in integrating multimodal paralinguistic data into a chat bot application [130]. Because of the multimodal data, it is difficult to decide how to represent the external data to the language model. It is also difficult to train a model to the encode the data without access to internal model features often hidden behind an API. Another obstacle is the lack of labeled multimodal datasets to build general representations. These black-box controller methods address all of these issues by training a model to encode external data based on a reward base signal.

### 7.5.1 Broader impacts and ethical considerations

The methods contained in this paper are intended to reliably, effectively, and easily influence open and closed-source LLMs. Abuse of these methods may result in negative social impacts and result in harm. Already prone to harmful bias [98], these methods may be used to exacerbate the generation of offensive text.

To mitigate risks of misuse, we recommend that researchers training black-box controllers on new tasks refrain from developing models explicitly aimed at generating harmful text. If such controllers are developed, we recommend that distributors not host such models. Researchers interested in safeguarding against bad actors using this technology may explore methods for identifying potentially nefarious prefixes and prompts. Toxicity classifiers such as the Perspective API may also be used to filter offensive generations.

In the course of developing these methods, additional compute resources were used in failed experiments and hyperparameter tuning.

## 7.6 Conclusion

As the primary means of interaction with LLMs shifts from local to remote connection, access to fundamental model features such as token embeddings, hidden states, and output probabilities have become restricted. Limiting access to model features directly impacts the viability of many methods for interacting with and controlling LLMs and prompt engineering methods require adaptation to CTG tasks. To address this, we propose methods for guiding the generated text of a base language model by way of a black-box controller. Unlike established CTG methods, the controller does not rely on any internal model features. Instead, the controller is trained using reinforcement learning to maximize rewards computed by an attribute discriminator.

The controller is evaluated on sentiment control and toxicity avoidance tasks. Results show that the black-box controller performs comparably to CTG baselines that rely on varying degrees of access to internal model features. Its ability to achieve strong performance despite operating under stricter access constraints—reflecting real-world user limitations—highlights the promise of black-box CTG methods as a viable and accessible alternative.

## 7.7 Limitations

These experiments do not evaluate the effect the black-box controller has on the topic of generated text when topic is not the control attribute. In our experience, the topic of the continuation may be affected by the prefix when the prompt is topically neutral.

A controller trained for a particular model is not expected to control another model. For now, a controller must be trained on a single task for a single base language model.

In terms of computational efficiency, the methods for training the black-box controller scale linearly with respect to dataset size. The black-box controller generates a fixed length prefix in constant time. Calculating the reward, where the base language model generates a fixed length continuation that is evaluated by the reward model, is computed in constant time. Calculating the gradient for and updating the controller model parameters occurs in constant time with respect to the size of the dataset.

# Chapter 8

# Conclusion

Conclusion goes here.

## References


[1] Margareta Ackerman, Ashok Goel, Colin G. Johnson, Anna Jordanous, Carlos León, Rafael Pérez y Pérez, Hannu Toivonen, and Dan Ventura. Teaching computational creativity. In *Proceedings of the International Conference on Computational Creativity*, pages 9–16. Association for Computational Creativity, 2017.

[2] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

[3] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL `https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf`.

[4] Margaret Boden. *The Creative Mind*. Abacus, 1992.

[5] Michele Boggia, Sardana Ivanova, Simo Linkola, Anna Kantosalo, and Hannu Toivonen. One line at a time — generation and internal evaluation of interactive poetry. In *Proceedings of the International Conference on Computational Creativity*, pages 7–11. Association for Computational Creativity, 2022.

[6] Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the Conference of the Association for Computational Linguistics*, volume 1, pages 4762–4779, 2019.

[7] Oliver Bown. Empirically grounding the evaluation of creative systems: Incorporating interaction design. In *Proceedings of the International Conference on Computational Creativity*, page 112–119, 2014.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya

Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages Pages 1877–1901. Curran Associates Inc., 2020. ISBN 9781713829546.

[10] Michael J. Cafarella, Alon Halevy, and Jayant Madhavan. Structured data on the web. *Communications of the ACM*, 54(2):72–79, 2011.

[11] Alex Calderwood, Vivian Qiu, Katy Ilonka Gero, and Lydia B. Chilton. How novelists use generative language models: An exploratory user study. In *Joint Proceedings of the Workshops on Human-AI Co-Creation with Generative Models and User-Aware Conversational Agents*. CEUR-WS, 2020.

[12] Filippo Carnovalini, Nicholas Harley, Steven T. Homer, Antonio Roda, and Geraint A. Wiggins. Meta-evaluating quantitative internal evaluation: A practical approach for developers. In *Proceedings of the International Conference on Computational Creativity*, page 213–217. Association for Computational Creativity, 2021.

[13] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *Proceedings of the Anniversary Meeting of the Association for Computational Linguistics*, pages 789–797. Association for Computational Linguistics, 2008. URL `https://aclanthology.org/P08-1090/`.

[14] Gianna Chien. Generating six-word stories. `http://cs230.stanford.edu/projects_fall_2020/reports/55790134.pdf`, 2020.

[15] Vladimír Chvátil. Codenames, 2015.

[16] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL `https://arxiv.org/abs/2110.14168`.

[17] S. Colton, J. Charnley, and A. Pease. Computational creativity theory: The face and idea descriptive models. In *Proceedings of the International Conference on Computational Creativity*, page 90–95, 2011.

[18] Simon Colton. Creativity versus the perception of creativity in computational systems. In *AAAI Spring Symposium: Creative Intelligent Systems*, volume 8, pages 14–20. AAAI, 2008. URL `http://www.aaai.org/Library/Symposia/Spring/2008/ss08-03-003.php`.

[19] Eugenio Concepción, Pablo Gervás, and Gonzalo Méndez. Evolving the INES story generation system: From single to multiple plot lines. In *Proceedings of the International Conference on Computational Creativity*, pages 220–227. Association for Computational Creativity, 2019. ISBN 978-989-54160-1-1. URL `http://computationalcreativity.net/iccc2019/papers/iccc19-paper-23.pdf`.

[20] Czech Games Edition. Codenames rules. https://czechgames.com/files/rules/codenames-rules-en.pdf, 2015. Official rulebook for the Codenames board game.

[21] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1edEyBKDS`.

[22] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang,

Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. https://arxiv.org/abs/2501.12948, 2025.

[23] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.222. URL `https://aclanthology.org/2022.emnlp-main.222`.

[24] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.222. URL `https://aclanthology.org/2022.emnlp-main.222`.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 4171–4186, 2019.

[26] Vincenzo Fragapane and Bernhard Standl. Work in progress: Creative coding and computer science education – from approach to concept. In *Proceedings of the IEEE Global Engineering Education Conference*, page 1233–1236, 2021.

[27] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 3356–3369. Association for Computational Linguistics, 2020. URL `https://aclanthology.org/2020.findings-emnlp.301`.

[28] Sean Glatch. How to write flash fiction stories. https://writers.com/how-to-write-flash-fiction, 2024.

[29] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

[30] Kazjon Grace and Mary Lou Maher. What to expect when you're expecting: The role of unexpectedness in computationally evaluating creativity. In *Proceedings of the International Conference on Computational Creativity*, pages 120–128. Association for Computational Creativity, 2014.

[31] Jacob Grimm and Wilhelm Grimm. Hänsel und gretel. In *Kinder- und Hausmärchen*, volume 1. Realschulbuchhandlung, Berlin, 1812. URL `https://en.wikipedia.org/wiki/Hansel_and_Gretel`. KHM 15.

[32] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[33] Laura Hanu and the Unitary team. Detoxify, 2020. URL `https://github.com/unitaryai/detoxify`.

[34] Mika Hämäläinen and Khalid Alnajjar. Modelling the socialization of creative agents in a master-apprentice setting: The case of movie title puns. In *Proceedings of the International Conference on Computational Creativity*, pages 266–273.

Association for Computational Creativity, 2019. ISBN 978-989-54160-1-1. URL http://computationalcreativity.net/iccc2019/papers/iccc19-paper-12.pdf.

[35] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.

[36] Michael N Jones, Jon Willits, Simon Dennis, and Michael Jones. Models of semantic memory. In Jerome R. Busemeyer, Zheng Wang, James T. Townsend, and Ami Eidels, editors, *Oxford Handbook of Mathematical and Computational Psychology*, pages 232–254. Oxford University Press, New York, 2015.

[37] Anna Jordanous. A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computation*, 4:246–279, 2012. doi: 10.1007/s12559-012-9156-1.

[38] Daniel Kahneman. *Thinking, Fast and Slow*. Macmillan, 2011.

[39] P. Kakavas. Computational thinking and creativity in K-6 education. *Formamente*, 14 (2):93–108, 2019.

[40] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *CoRR*, abs/1909.05858, 2019. URL http://arxiv.org/abs/1909.05858.

[41] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2017. URL https://arxiv.org/abs/1412.6980.

[43] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. GeDi: Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 4929–4952. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.findings-emnlp.424.

[44] Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. Sage, 3rd edition, 2013.

[45] John E Laird. *The Soar Cognitive Architecture*. MIT press, 2012.

[46] Carolyn Lamb and Daniel G. Brown. Twitsong 3.0: Towards semantic revisions in computational poetry. In *Proceedings of the International Conference on Computational Creativity*, pages 212–219. Association for Computational Creativity, 2019. ISBN 978-989-54160-1-1. URL http://computationalcreativity.net/iccc2019/papers/iccc19-paper-9.pdf.

[47] Carolyn Lamb, Daniel G. Brown, and Charles Clarke. Human competence in creativity evaluation. In *Proceedings of the International Conference on Computational Creativity*, page 102–109. Brigham Young University, 2015.

[48] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

[49] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, volume 1, pages 4582–4597. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.acl-long.353.

[50] Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, volume 1, pages 6691–6706. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.acl-long.522.

[51] Angli Liu, Jingfei Du, and Veselin Stoyanov. Knowledge-augmented language model and its application to unsupervised named-entity recognition. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1142–1150, 2019. URL https://doi.org/10.18653/v1/n19-1117.

[52] Angli Liu, Jingfei Du, and Veselin Stoyanov. Knowledge-augmented language model and its application to unsupervised named-entity recognition. In Jill Burstein, Christy

Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 1142–1150. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1117. URL https://doi.org/10.18653/v1/n19-1117.

[53] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):Article 195, 2023. ISSN 0360-0300. doi: 10.1145/3560815. URL https://doi.org/10.1145/3560815.

[54] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150. Association for Computational Linguistics, 2011. URL https://aclanthology.org/P11-1015.

[55] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *Proceedings of the International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

[56] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. https://arxiv.org/abs/1802.03426, 2020.

[57] Lennart Meincke, Ethan R. Mollick, Lilach Mollick, and Dan Shapiro. Prompting science report 1: Prompt engineering is complicated and contingent. Technical report, SSRN, 2025. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5165270. Available at SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5165270.

[58] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *Proceedings of the International Conference on Learning Representations*, 2017.

[59] Meta AI. Introducing Llama 3.1: Our most capable models to date, 2024. URL https://ai.meta.com/blog/meta-llama-3-1/. Accessed on October 15, 2024.

[60] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119, 2013.

[61] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, 2016.

[62] Eric Miller. An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology*, 25(1):15–19, 1998.

[63] Robert Morain, Kenneth Vargas, and Dan Ventura. Symbolic semantic memory in transformer language models. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 992–998, 2022. doi: 10.1109/ICMLA55696.2022.00166.

[64] Robert Morain, Branden Kinghorn, and Dan Ventura. Are language models unsupervised multi-domain CC systems? In *Proceedings of the International Conference on Computational Creativity*, pages 39–43. Association for Computational Creativity, 2023.

[65] Hugo G. Oliveira and Ana O. Alves. Poetry from concept maps–yet another adaptation of PoeTryMe's flexible architecture. In *Proceedings of the International Conference on Computational Creativity*, pages 246–253. Sony Computer Science Laboratories, June 2016. URL `http://www.computationalcreativity.net/iccc2016/wp-content/uploads/2016/01/Poetry-from-Concept-Maps.pdf`.

[66] OpenAI. Introducing ChatGPT. `https://openai.com/blog/chatgpt`, 2023. Accessed 2023-4-11.

[67] OpenAI. ChatGPT (GPT-4o version). `https://chat.openai.com`, 2024. Accessed 2025-2-14.

[68] OpenAI. Learning to reason with LLMs. Blog post, 2024. URL `https://openai.com/index/learning-to-reason-with-llms/`. Accessed: February 26, 2025.

[69] OpenAI. Prompt engineering. `https://platform.openai.com/docs/guides/prompt-engineering`, 2025. Accessed: February 27, 2025.

[70] Korawit Orkphol and Wu Yang. Word sense disambiguation using cosine similarity collaborates with Word2vec and WordNet. *Future Internet*, 11(5):114, 2019.

[71] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.

[72] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf`.

[73] A. Pease and S. Colton. Computational creativity theory: Inspirations behind the FACE and the IDEA models. In *Proceedings of the International Conference on Computational Creativity*, page 72–77, 2011.

[74] Alison Pease and Simon Colton. On impact and evaluation in computational creativity: A discussion of the Turing test and an alternative proposal. In *Proceedings of the Artificial Intelligence and the Simulation of Behaviour Symposium on Computing and Philosophy*, pages 15–22, 2011.

[75] Max Peeperkorn, Dan Brown, and Anna Jordanous. On characterizations of large language models and creativity evaluation. In *Proceedings of the International Conference on Computational Creativity*, page 143–147. Association for Computational Creativity, 2023.

[76] Max Peeperkorn, Tom Kouwenhoven, Dan Brown, and Anna Jordanous. Is temperature the creativity parameter of large language models? In *Proceedings of the International Conference on Computational Creativity*. Association for Computational Creativity, 2024.

[77] Rafael Pérez Y Pérez and Mike Sharples. MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(2):119–139, 2001.

[78] Rafael Pérez y Pérez. *MEXICA: a computer model of creativity in writing.* PhD thesis, University of Sussex, 1999.

[79] Rafael Pérez y Pérez and Mike Sharples. *An introduction to narrative generators: how computers create works of fiction.* Oxford University Press, 2023.

[80] Charles Perrault. Little red riding-hood. In *Tales of Past Times Written for Children*, pages 6–8. E.P. Dutton and Co., New York, 1923. URL `https://www.colorado.edu/projects/fairy-tales/charles-perrault-little-red-riding-hood`.

[81] Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz an Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 43–54, 2019. URL `https://doi.org/10.18653/v1/D19-1005`.

[82] Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, pages 43–54. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1005.

[83] Todd Pickering and Anna Jordanous. Applying narrative theory to aid unexpectedness in a story generation system. In *Proceedings of the International Conference on Computational Creativity*, pages 213–220. Association for Computational Creativity, 2017.

[84] PromptBase. Promptbase: Marketplace for AI prompts. https://promptbase.com/, 2025. Accessed: March 1, 2025.

[85] Jing Qian, Li Dong, Yelong Shen, Furu Wei, and Weizhu Chen. Controllable natural language generation with contrastive prefixes. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2912–2924, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.229. URL `https://aclanthology.org/2022.findings-acl.229`.

[86] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learn-

ers. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf, 2018. Accessed 2023-4-11.

[87] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[88] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[89] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the International Conference on Machine Learning*, volume 139, pages 8821–8831, 2021.

[90] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019.

[91] Graeme Ritchie. The JAPE riddle generator: technical specification. Technical Report EDI-INF-RR-0158, School of Informatics, University of Edinburgh, 2003.

[92] Graeme Ritchie. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines*, 17:76–99, 2007.

[93] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. URL http://arxiv.org/abs/1910.01108.

[94] Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. Atomic: An atlas of machine commonsense for if-then reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3027–3035, 2019.

[95] Gal Sasson Lazovsky, Tuval Raz, and Yoed N. Kenett. The art of creative inquiry—from question asking to prompt engineering. *The Journal of Creative Behavior*, 59(1):e671, 2025.

[96] Piotr Sawicki, Marek Grzes, Luis Fabricio Góes, Dan Brown, Max Peeperkorn, Aisha Khatun, and Simona Paraskevopoulou. On the power of special-purpose GPT models to create and evaluate new poetry in old styles. In *Proceedings of the International Conference on Computational Creativity*, pages 10–19. Association for Computational Creativity, 2023.

[97] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `https://arxiv.org/abs/1707.06347`.

[98] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. The woman worked as a babysitter: On biases in language generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, pages 3407–3412. Association for Computational Linguistics, 2019. URL `https://aclanthology.org/D19-1339`.

[99] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from Language models with automatically generated prompts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4222–4235. Association for Computational Linguistics, 2020. URL `https://aclanthology.org/2020.emnlp-main.346`.

[100] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013. URL `https://aclanthology.org/D13-1170`.

[101] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4444–4451, 2017.

[102] Brad Spendlove and Dan Ventura. Creating six-word stories via inferred linguistic and semantic formats. In *Proceedings of the International Conference on Computational Creativity*, pages 123–130. Association for Computational Creativity, 2020.

[103] Brad Spendlove and Dan Ventura. Humans in the black box: A new paradigm for evaluating the design of creative systems. In *Proceedings of the International Conference on Computational Creativity*, pages 311–318. Association for Computational Creativity, 2020.

[104] Brad Spendlove and Dan Ventura. Competitive language games as creative tasks with well-defined goals. In *Proceedings of the International Conference on Computational Creativity*, pages 291–299. Association for Computational Creativity, 2022.

[105] Brad Spendlove and Dan Ventura. A constraint-centric accounting of some aspects of creativity. In *Proceedings of the International Conference on Computational Creativity*, page 332–336. Association for Computational Creativity, 2023.

[106] Brad Spendlove, Nathan Zabriskie, and Dan Ventura. An HBPL-based approach to the creation of six-word stories. In *Proceedings of the International Conference on Computational Creativity*, pages 161–168. Association for Computational Creativity, 2018.

[107] Flora Annie Steel. Jack and the beanstalk. In *English Fairy Tales*, pages 136–153. Macmillan and Co., Limited, London, 1918. URL `https://www.gutenberg.org/files/17034/17034-h/17034-h.htm`.

[108] Flora Annie Steel. The story of the three bears. In *English Fairy Tales*. Macmillan and Co., 1922. URL `https://www.gutenberg.org/ebooks/17034`.

[109] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics*, pages 13003–13051. Association for Computational Linguistics, 2023.

[110] Joe Toplyn. Witscript: A system for generating improvised jokes in a conversation. In *Proceedings of the International Conference on Computational Creativity*, pages 22–31. Association for Computational Creativity, 2021.

[111] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. URL `https://arxiv.org/abs/2302.13971`.

[112] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich,

Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. https://arxiv.org/abs/2307.09288, 2023.

[113] Endel Tulving. *Episodic and Semantic Memory*. Academic Press, Oxford, England, 1972.

[114] Bradley Tyler, Katherine Wilsdon, and Paul Bodily. Computational humor: Automated pun generation. In *Proceedings of the International Conference on Computational Creativity*, pages 181–184. Association for Computational Creativity, 2020. ISBN 978-989-54160-2-8. URL `http://computationalcreativity.net/iccc20/papers/152-iccc20.pdf`.

[115] Josep Valls-Vargas, Jichen Zhu, and Santiago Ontañón. Towards automatically extracting story graphs from natural language stories. In *AAAI Workshops*, 2017.

[116] Tony Veale. From symbolic caterpillars to stochastic butterflies: Case studies in re-implementing creative systems with LLMs. In *Proceedings of the International Conference on Computational Creativity*, pages 236–244. Association for Computational Creativity, 2024.

[117] Dan Ventura. Mere generation: Essential barometer or dated concept. In *Proceedings of the International Conference on Computational Creativity*, pages 17–24. Sony CSL, 2016.

[118] Dan Ventura. How to build a CC system. In *Proceedings of the 8th International Conference on Computational Creativity*, pages 253–260. Association for Computational Creativity, 2017.

[119] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`, 2020.

[120] Denny Vrande. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[121] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32*, pages 3261–3275, 2019.

[122] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021.

[123] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, pages 24824–24837. Curran Associates Inc., 2022. ISBN 9781713871088.

[124] G.. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowlege-Based Systems*, 19:449–458, 2006.

[125] Geraint A. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge Based Systems*, 19(7):449–458, 2006. doi: 10.1016/j.knosys.2006.04.009. URL `https://doi.org/10.1016/j.knosys.2006.04.009`.

[126] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL `http://arxiv.org/abs/1910.03771`.

[127] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, 2020. URL `https://aclanthology.org/2020.emnlp-demos.6`.

[128] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, V.G.Vinod Vydiswaran, and Hao Ma. IDPG: An instance-dependent prompt generation method. In *Proceedings of the Conference of the North American Chapter of the Association for Computational*

*Linguistics: Human Language Technologies*, pages 5507–5521. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.naacl-main.403.

[129] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *Proceedings of the International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

[130] Weiwei Yang, Spencer Fowers, David Tittsworth, Amber Hoak, Thiago Vallin Spina, Kate Lytvynets, Christopher O'Dowd, Andrea Britto Mattos Lima, Whitney Hudson, Ben Cutler, Prachi Patel, and Robert Morain. Multimodal paralinguistic prompting for large language models. https://www.microsoft.com/en-us/research/project/project-rumi/, 2023. Accessed 2024-2-13.

[131] M. Yee-King, A. Fiorucci, and M. d'Inverno. Designing an AI-creativity music course. In *Proceedings of The International Conference on AI and Musical Creativity*, 2024.

[132] Hanqing Zhang and Dawei Song. DisCup: Discriminator cooperative unlikelihood prompt-tuning for controllable text generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3392–3406. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.emnlp-main.223.

[133] Hanqing Zhang, Sun Si, Haiming Wu, and Dawei Song. Controllable text generation with residual memory transformer. *CoRR*, abs/2309.16231, 2023. URL https://arxiv.org/abs/2309.16231.

[134] Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. A survey of controllable text generation using transformer-based pre-trained language models. *Association for Computing Machinery Computing Surveys*, 56(3), 2023. ISSN 0360-0300. URL https://doi.org/10.1145/3617680.

[135] Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. Differentiable prompt makes pre-trained language models better few-shot learners. *arXiv preprint arXiv:2108.13161*, 2021.

[136] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul F. Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *CoRR*, abs/1909.08593, 2019. URL http://arxiv.org/abs/1909.08593.