KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Enforcing creative constraints in autoregressive language models during generation for musical parodies

Anton Lintermans

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotor:
Prof. dr. ir. Luc De Raedt

Evaluatoren:
Ir. W. Eetveel
W. Eetrest

Begeleider:
Dr. ir. T. Winters

Academiejaar 2023 – 2024

Preface

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my wive and the rest of my family.[TODO]

Anton Lintermans

Contents

Abstract

The `abstract` environment contains a more extensive overview of the work. But it should be limited to one page. [TODO]

Samenvatting

In dit `abstract` environment wordt een al dan niet uitgebreide Nederlandse samenvatting van het werk gegeven. Wanneer de tekst voor een Nederlandstalige master in het Engels wordt geschreven, wordt hier normaal een uitgebreide samenvatting verwacht, bijvoorbeeld een tiental bladzijden. (Volgens Lenny moet het 1 a 3 bladzijden zijn) [TODO]

Introductie

Achtergrond

Gerelateerd werk

Onderzoeksvraag

Aanpak

Evaluatie

Conclusie

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
POS	Part-of-speech
RNN	Recursive Neural Network
LSTM	Long Short Term Memory

Symbols

Chapter 1

Introduction

With the arrival of ChatGPT [31], the interest in Artificial Intelligence and Machine Learning has never been higher. It has found its way into almost every field, from journalism, research to marketing , education , and beyond. Amidst this widespread adoption across various domains, this thesis focuses on a particular niche. It delves into the creative possibilities of AI, particularly in creative text generation. Most large language models like ChatGPT, are already very capable of generating creative text like poems, song lyrics, parodies, ... Although it outputs qualitative and coherent text, it does not always comply with the rules associated with the generated poem or parody. This can be seen in fig. 1.1 below, where ChatGPT was asked to write a haiku about how crazy pineapple on a pizza is. The result unfortunately is not a correct haiku, as it did not adhere to the 5-7-5 syllable rule. The generated poem comes close but has a 5-6-5[1] syllable structure instead[2]. When the same prompt is given again to the chatbot (see appendix A), a new haiku is generated that does adhere to the syllable rule. This shows that the model does have a notion about the rules associated with haiku and tries to meet them but does not always succeed. Therefore this thesis looks into enforcing these creative constraints or rules during generation to ensure they are (almost) always fulfilled. The focus will not be on generating poems, but on generating musical parodies with very similar constraints as poems but are more strictly defined.

Golden waves of heat
Pineapple dances wild
Pizza's sweet embraced

FIGURE 1.1: Haiku about how crazy pineapple on a pizza is
Generated by ChatGPT 3.5 Turbo[31] on 12/12/2023

[1]The number of syllables of the poem was counted by using syllablecounter.net

[2]When tested on other chatbots like Bard [11], similar results are achieved which can be found in appendix A

1.1 Context

Musical parody generators using language models have been researched before, but not to a large extent. One of the most notable works in this field is "Weird AI Yankovic: Generating Parody Lyrics", by M. Riedl [47], which can generate parodies based on the original lyrics and a prompt to indicate the subject of the parody. It combines two complementing pre-existing language models and extra added logic to create a parody with the same syllable counts and rhyming scheme as the original song. However, this approach, while innovative, highlights the struggle to create coherent and contextually relevant text that resembles a qualitative lyric. Furthermore, many existing systems in the literature (see chap. 3), including the one from Riedl, try to maintain the structure of the original songs but often fail to take into account the original song's thematic and stylistic elements, leading to parodies that lack resonance with the source material.

A critical gap identified in the existing literature for generating musical parodies and poetry as well (see chap. 3) is the use of models that, by today's standards, could be considered out-of-date. Many researchers use models such as GPT-2 model [43] from OpenAI or the T5 model [44] from Google, which were released more than four years ago. When compared to more recent models, they greatly fall behind in areas such as performance, and input size[3]. Also, current methods for parody or poem generation often have a tailor-made solution for their specific problem, which makes it very difficult to adapt it to add new constraints or use other models. All this shows that the field of musical parody generation still presents numerous opportunities for improvement and refinement in its methodologies and applications.

1.2 Research Question

Addressing these limitations, this thesis will mainly research how creative constraints, such as exact syllable amounts, and certain rhymings, can be enforced on language models during generation. More specifically state-of-the-art autoregressive language models are used, where the next part of a sentence is predicted based on the past, the sequence of words that came before. The focus is on generating musical parodies since its creative constraints can be strictly defined and the research about it has been limited. However, the aim is to build a method agnostic to which constraints or language models are used. This way the research could be used for other creative fields, such as generating different kinds of poems.

[3]Which can be seen on the Open LLM Leaderboard from Huggingface, where GPT2 (The best model of the two according to M. Reidl [47]) scores only an average of 28.8, while the top models at the time of writing score more than double with a score of 79. GPT-2 only has an input of 1024 while the newest models offer a much larger input size with Llama 2 having a maximum input size of 4096 tokens [52] and the newest Gemini 1.5 model [36] having a maximum of up to one million tokens.

1.3 Approach

To answer the above question, two different approaches are explored to generate musical parodies. Both approaches involve using a generic generating framework, to which specific constraints for parodies are added. The first method uses the constrained language model to generate the song line per line, while the second method generates the entire song with only one prompt. The used constraints are for both methods theoretically the same but require a different implementation for each approach. Both have their advantages and disadvantages. The first method has more control over the generation per line and uses far less memory, but the second approach does allow for better overall text coherence because it is generated in one go.

1.4 Structure

The structure of this thesis contains seven chapters. After this introduction, a background chapter, Chapter 2, follows with a detailed explanation of musical parodies, the basis of language models, and how these models are used to generate text. Chapter 3 delves into the existing methods for both generating musical parodies and poems. The chapter also goes broader and looks at different techniques for enforcing different kinds of constraints in general. Next, Chapter 4, states the specific research question that this thesis aims to address. Chapter 5 then presents a proposed approach to solve the problem statement. This approach is next evaluated in Chapter 6 with both an automatic and human evaluation. Finally, Chapter 7 will conclude this thesis.

1.5 Acknowledgements

Use of GAI

The thesis was written with the help of GAI. Grammerly [1] was used to correct any spelling mistakes and rewrite small parts of a sentence. ChatGPT (both ChatGPT 4 [32] and ChatGPT 3.5 [31]) was utilized to rewrite parts of paragraphs. The author confirmed no new information was added, that was not originally given.

Resources

The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government - department EWI.

Chapter 2

Background

[TODO] Explanation of logitss POS tagging uitleggen

2.1 What is a parody lyrics

2.2 Language Models

2.2.1 Basic Language Models

2.2.2 Transformer Models

2.2.3 Large Language Models

2.2.4 Pre-Trained Models

2.3 HuggingFace

2.3.1 Transformers Library

2.3.2 Huggingface Models Library

2.4 POS Tagging

2.5 Search Methods for Language Models

2.6 Backtracking

Chapter 3

Related Literature

This chapter discusses existing approaches to generating parody songs, focusing mainly on Transformer models. Following this, various strategies for constrained text generation are discussed, and the final section covers different Backtracking and search algorithms for generative text.

3.1 Parody Lyrics Generation

Over the last decade, there have been significant advancements in artificial intelligence and machine learning, but these advances have not been fully utilized in the field of parody lyrics generation. Only a few research papers have introduced ML techniques to generate parodies based on existing lyrics. This section will mainly focus on those papers that use (auto-regressive) transformer models, but other techniques will also be briefly presented for the sake of completeness.

3.1.1 Transformer Based Techniques

Only one paper stands out, that uses transformer models and comes somewhat close to what the research of this thesis is about. That paper is written by Mark Riedl and is called *"Weird AI Yankovic: Generating Parody Lyrics"* [47]. He introduces a system that uses two pre-trained auto-regressive transformer models, GPT-2 and XLNET, introduced in [42] and [57], respectively. With the help of those models, the system generates a new lyric that adheres to the original song's syllable count and rhyme scheme, with content related to the context prompt.

The user inputs the syllable count and rhyming identifier for each segment and specifies whether the segment should end with a period. Each line consists of one or more segments, depending on the rhyming scheme. Segments with the same rhyming identifier will rhyme on their last word. In case of interior rhymes, some lines will have multiple segments. Additionally, the user must provide a context prompt that steers the topic of the parody. Although the paper by Riedl does not mention how these parameters can be inferred, an implementation can be found on the following Github repository [48] created by Riedl, which includes a full implementation of the algorithm. The given code can only detect rhymes if they occur at the end of the line and thus can not detect interior rhymes.

The system uses two constraints to create parodies: it maintains the same rhyming scheme and syllable count per line. These restrictions are only upheld if they can be accurately deduced from the original lyrics by the user. When a rhyming word is needed, the system looks it up in its near-rhyme dictionary to find possible matches. It then uses GPT-2 to generate m lines of length n, given the context as a prompt. The rhyming words are ranked by their average logit of each position on all m lines. This ensures that the selected rhyme is closely related to the original prompt since it has the highest probability of being generated. The syllable count is managed by using XLNet for backward generation, starting from the picked rhyming word and adding masks until the target syllable count is reached. For lines without specific rhymes, GPT-2 is used in forward generation, creating one token at a time and stopping once the required syllable count is met. If the syllable count is exceeded, the line is discarded. The system generates multiple possibilities per line and selects the one with the highest posterior probability when run through GPT-2.

Although the system can produce parody lyrics and adhere to the syllable and rhyming constraints, it is far from perfect. The generated lyrics are not coherent text that resembles actual lyrics. For instance, in the generated parody of *My Shot* (example in the paper [47]) the line *"And unreasonable of the just honorable"* is followed by *"Dishonorable imposition on the honourable."*. In the authors' opinion, these two lines do not make much sense together. Even when lines do make sense they do not resemble lyric lines but are more like sentences in a story, which can be seen with the lines *"In our hearts it is only a little more"*, *"In our hearts it is only a little more"* and *"In our hearts it is only a little more"* from the same example. This is partially because GPT-2 and XLNET are not trained to generate lyrics but rather to write stories. Although these models can be fine-tuned on lyrics, like Andersson did [3], it only slightly improves the generation. Both Lau [18] and Rodrigues et al. [49] fine-tuned GPT-2 on generic song lyrics generation using an umbrella of different genres, but the generated songs still were not adequate. The main reason why the generated lyrics are of poor quality has more to do with the models being used. Compared to the models that are released almost every day now, the two models can be called *ancient*. They are not trained to do anything other than generate text and a story but are not trained to do tasks when instructions are given. It is also worth noting that the system does not take the original lyrics as input; therefore, all resemblance to the original song is lost.

Chang et al. [7] also use GPT-2, their aim was not to create parodies but to transfer the style of the original song to a new style. Since the structure of the song and the rhyming scheme stay more or less the same the output can also be seen as a parody. The user must input a key sentence or prompt to guide the style or topic of the new lyric. The system then creates a new song with a lyrics fine-tuned GPT-2 model based on the prompt, and then it analyzes both the original and the created song for word relation dependencies. Similar relations found in the created lyric are then replaced in the original one. Afterward, the words that need to rhyme, are replaced by synonyms that rhyme with the original word. Compared to Riedl, it has more coherent lyrics, but this is because much more of the original structure including the relational dependencies is maintained, which automatically results in a more coherent text. But unlike Riedl it does not take the syllable constraint into account.

Ram et al. [46] took a different approach. They developed a collaborative system that supports songwriters in writing lyrics. The system is based on the T5 model created by Google in 2020, which is described in more detail in this paper [45]. Unlike other systems that generate complete lyrics or parodies, this system generates 1-5 lines at a time. It allows songwriters to specify the number of syllables per line, whether to end on a specific word or a word that rhymes with a given word. It is not made to create parodies but can be used to create parts of parody lyrics by giving the rhyming words and exact syllable counts. Instead of hard-coding these constraints, the model is trained to adhere to these constraints. This is done by using tags like *'[RHYME]'* to indicate the rhyming word and *'[LINE]'* to indicate a new line. The number of syllables is given by *'syllable count: 5'*. These tags/commands were included when the T5-model was fine-tuned with lyrics. Some songwriters tested the system and were happy with the results, but they wished there was a way to indicate the mood, style, or tone.

The English language is the main focus of this thesis, but it is worth mentioning some techniques used in the Chinese language. Chipsong, introduced by Liu et al. [21] and DeepRapper, by Xue et al. [55], are auto-regressive transformer-based systems that generate pop song lyrics for short videos and rap lyrics respectively in the Chinese language. The transformer models were both built from the ground up and were trained according to their needs. Chipsong focuses on controlling the length of words and sequences, which correspond to syllable count and sentence length control. Each Chinese word is made up of one or more characters and each character corresponds to one syllable. Chipsong adds a special Begin-Internal-End (BIE) word-granularity embedding sequence and a guided attention mechanism to the transformer model, to achieve this control. Deeprapper does not have any word or syllable control but focuses on the rhyming aspect and incorporates the rhythm structure, by adding a *[BEAT]* tag, to indicate when a beat occurs. Both models are trained with reverse order generating to better accommodate the rhyming. This way they first start by picking the rhyme word, which needs to be placed at the end, and then generate starting from the picked word. Deeprapper goes one step further and allows for rhyming on multiple consecutive tokens at the end of a line.

3.1.2 Other ML Methods

Although the thesis is focused on auto-regressive transformer models, a quick overview of parody lyrics generation with four other ML techniques follows.

Word Embeddings

Not only language models can be used to create parodies, but also word embeddings which are vector representations of words can be used. Two popular word embeddings are Word2Vec [27] and GloVe [35]. Oliveira used GloVe to create WeirdAnalogyMatic [29], a word embedding system that can transfer the theme of a lyric to a new one. The theme of a song is captured in its title and is represented by the average embedding vectors of the words in the title. Then, new replacement words for the lyrics are found by subtracting the vector of the old theme from the embedding vector of the original one and adding the embedding representing the new theme. The new word is then the word, which embedding vector is the closest to the generated one. If this replacement is done on all words, weird and incoherent text would appear, therefore some constraints were applied. Stopwords are not replaced, a replacement history is kept to not use the same words, the replacement word has the same POS tag and syllable count, and the replaced word rhymes with the original one if needed. The nearest word to the new embedding vector, that meets the constraints will be chosen. If the word is too far from the vector, the original word is kept. The best results were achieved by requiring all constraints except for the rhyming one, as most words were not being replaced because a valid replacement was often too far from the new vector.

BERT

Following his work with word embeddings Oliveira made something similar, but now by using a masked language model, discussed in [30]. He uses the BERT language model, which is a well-known model introduced in [10]. The objective of this technique is to shift the theme towards a new one, just like WeirdAnalogyMaic. However, instead of using word embeddings, a masked language model is employed which can mask words, allowing them to be replaced. Normally the word is selected that BERT gives the highest probability for that mask, but now the word is chosen that also adheres to the given constraints. As before stop words stay, only content words are replaced. There are four main constraints; the word must have the same amount of syllables, it has to have the same POS tag, and also the same inflection. To keep the same rhyming regime, it makes sure that the word at the end of each line rhymes with the original word. Sometimes rhyming words are not found, then the original is kept. Finally, to be able to shift the theme, the word that is most similar to the given theme is chosen, while still adhering to the constraints.

To Sing Like a Mockingbird

Gatti et al. have developed a system called *To Sing Like a Mockingbird*, which can generate musical parodies based on a given news article. The system uses the headline and the summary of the article to create a corpus of characteristic and infrequent words. The summary and headline will be lemmatized and POS tagged using the coreNLP library [25], and irrelevant words such as stop words and frequent words are discarded. The corpus is then expanded with relevant words and synonyms from WordNet, the Oxford Thesaurus, and Wikidata. The system then replaces words from the original lyrics with words from the generated corpus, but only from the most memorable part of the song, which is usually the chorus. As with other techniques, only the content words are replaced, and the replacement word must have the same POS tag and number of syllables as the original word. If the replaced word is at the end of a line, it must also (near-)rhyme with the original word. If multiple substitutions are possible, a language model is used to see which is the most probable. If no substitution is possible, the original stays. The presented system presents promising results, but only a small part of the song is changed.

LSTM

Before Transformer models were developed, other techniques were used to try to generate lyrics. One of the popular architectures was LSTM (Long Short Term Memory), which Potash et al. used to create a language model named *Ghostwriter* [40]. The model is trained to generate rap lyrics. To ensure that the model understood the structure of lyrics, two tags were included in training: *<endLine>* and *<endVerse>*, to indicate the end of a line and a verse, respectively. This helps the model generate lyrics with a familiar lyric structure and create end-of-line rhymes.

3.2 Poetry Generation

Poetry has very similar constraints to parody lyrics, therefore the following section discusses different relevant studies on generating poetry by enforcing poetic constraints with transformer models. Some notable techniques that do not use transformers are also given below.

3.2.1 Transformer Approaches

SongNet

Li et al. propose SongNet [19] a tranformer-based framework that can generate controlled format text, specifically poems. The framework ensures that the generated text adheres to a pre-specified format and a given rhyming scheme while maintaining sentence integrity. This is achieved by adding special tokens to the input that specify the format and rhyming scheme. Additionally, they add special embeddings into the transformer architecture to understand these symbols and force the generation to adhere to them. The model is first trained on a general corpus of English or Chinese text and then fine-tuned with specific poetry. The effectiveness of the framework was tested on English sonnets and Chinese SongCi, and the results showed its capabilities. The SongCi model outperformed the sonnets model, possibly due to the larger training set of the former.

ZEST

Tian and Peng introduce a new zero-shot method for generating sonnets called ZEST [51], which does not require any poetic data. This method generates the sonnet in four steps. First, it uses a fine-tuned T5 model that generates a certain number of keywords for each line of the sonnet when given a title. Second, for every line that rhymes with a previous line, ZEST generates a list of rhyming words according to the CMU dictionary along with the probability that the model would generate the word in step one. Third, ZEST makes the generated words more vivid by using two methods. The first method generates an imagery word given another word. They sample multiple keywords randomly and take the most probable one. The second method generates simile phrases for given adjectives. Finally, in the last step, ZEST generates the sonnet given the previously generated words. A GPT-NEO model fine-tuned with the same news and stories dataset as the T5 model was fine-tuned with, is used but in a reverse manner from right to left. This way, the method generates the sonnet line-by-line, starting with the rhyme word and making its way to the front of the line. To ensure the keywords are included, an adapted version of the Grid Beam Search algorithm[12] is used, which also takes the required rhythm into account. When tested against other poetry generators ZEST outperforms them according to both automatic and human evaluations. Due to its planning scheme, the generated sonnets are much more coherent, not only on a sentence level but also globally.

PoeLM

Ormazabal et al. present PoeLM [33], a transformer-based model to generate user-specified poems. The user defines the structure of the poem using a markup language-like format in the prompt. The model is trained on general text and then fine-tuned on poems with annotations for rhyming, length, and included words. During inference, the model generates 3000 different versions, which are filtered based on various criteria such as the exact number of lines, syllables, included rhyming patterns, repetition of words, and the BLUE score. The best candidates are ranked and then evaluated for fluency once more by the model without the constraints added to the prompt. Finally, the poem with the highest probability is chosen.

CoPoet

Chakrabarty et al. present CoPoet [6], a new collaborative system that allows users to write poems interactively with the help of the T5 model fine-tuned specifically for poetry. The system is designed to generate one or two lines at a time, which adhere to certain constraints specified in the prompt. These constraints may include subject matter, rhyming words, and specific words to include in the poem. Unlike with PoeLM the structure and other constraints are specified in natural language. CoPoet outperforms a more generic trained instruction model and enables writers to collaboratively generate coherent and original poems to their liking.

GPoet-2

Lo et al. propose GPoeT-2[22], a system that can generate limericks with an AABBA rhyme scheme and without any initial phrase. Two GPT-2 models were fine-tuned to make this work. The first one is fine-tuned on regular limericks, while the second one is trained on limericks generated from right to left to better incorporate end rhyming. For optimal results, the first model produces the first line, while the right-to-left model generates the remaining four lines.

ByGPT5

Belouadi and Eger present ByGPT5 [5], a decoder-only transformer model that generates quatrains. This character-based model is first trained on English and German Wikipedia-like text and then fine-tuned using labeled quatrains that indicate the rhyme scheme. ByGPT5 outperforms other models, including the similarly fine-tuned GPT-2 model. The study also suggests that character-based models are more effective than subword-level models for generating character styles.

GPoeT

Popescu-Belis et al. introduce GPoet[38], a fine-tuned version of GPT-2 on poetry, which is capable of generating rhyming lines. Although it shares the name with GPoeT-2, discussed above, they are not related. Training the model solely on 3 million lines of natural poetry does not yield good results with only 11% of the generated lines having consecutive rhymes. To improve this, they propose training the GPT-2 model on synthetic data. This data is created using GPT-2 to create quatrains in the AABB or ABAB rhyming scheme line by line, with each line generated individually to control the number of syllables. The last word of the second rhyming line is replaced by a rhyming word using their own made phonemic rhyming dictionary. By fine-tuning GPT-2 on the created dataset with the AABB structure shows that it can generate rhyming lines 59% of the time. The ABAB structure is deemed to be more difficult and the model could only achieve an accuracy of 45%. This shows the effectiveness of using synthetic data but also shows its limit in that it can only be trained for one scheme.

3.2.2 RNN Approaches

Automatic Poetry Generation from Prosaic Text

Van de Cruys [53] presents a new method to generate poetry automatically. The method involves training an RNN that always has the previous sentence as input to ensure coherence over multiple sentences. The RNN is not trained on a poetic corpus, but rather on prosaic text from the internet. The system generates poetry by altering the probability outputs to adhere to two constraints: a rhyming constraint and a topical constraint. The rhyming constraint is incorporated by training the network to generate sentences in reverse, starting with the last word of the sentence that needs to rhyme. The probability is changed such that only the words that rhyme to the given word are able to be generated. To include the topical constraint, another model learns a probability distribution for the vocabulary given a certain topic. This distribution is combined with the original probability distribution from the RNN, but only when the original distribution has a high entropy. This ensures that the generated text is both coherent and fluent. Multiple candidate verses are generated, and the one with the best score is selected. The system has outperformed state-of-the-art systems and can generate coherent and fluent poetry.

CR-PO

Popescu-Balis et al. propose CR-PO [37] an interactive system that can generate French poems. The user has the option to choose one out of four formats, to which the system generates a poem that adheres to the selected format. Additionally, the user can modify the topic, emotion, or rhyme scheme to customize the poem according to their liking. The initial poem is generated by a generic character-based RNN autoregressive language model. This model is trained on a corpus of French poems and generates the poem line by line to adhere to the chosen scheme. After fulfilling 85% of a line, it tries to generate an ending within the length limit. If it fails, the system restarts at the 85% point and relaxes the heuristics each time until a satisfactory line is found. The selected topics and emotions are imposed by replacing those words that are least related. The words are ranked by two methods, either by fine-tuned LMs for each emotion/topic or by the occurrence of the word in the training data. The words are then replaced with a masked LM like CamemBERT or with the original or topic fine-tuned LM, but those take only the left context into account. Furthermore, the rhyming scheme is enforced by replacing the words that need to rhyme by generating candidates using a dictionary and the generic LM to pick the most likely word that also has the same POS tag as the original word. After testing it was noted that while the system produces coherent poems that adhere to all the specifications, the poems do not have a real story or meaning behind them.

3.2.3 LSTM Approaches

Rythmic Verse Generation

Hopkins and Kiela [13] introduce two different methods for generating rhythmic poetry. The first model uses a Long Short-Term Memory (LSTM) network to learn the phonetic patterns of the given data. The second model takes a different approach and uses a generic language model that is constrained by a weighted finite state machine to generate the desired poetry. This allows for generating unseen poetry formats without retraining the language model, which is not the case with the first method. The finite state machine controls the format of the poetry at the word level, acting as a classifier that accepts or rejects new words proposed by the language model. To enforce a specific theme the probability of words that are similar to a given theme is increased. This approach can also be used to include certain poetic devices in the generated poetry. Human evaluation revealed that the generated poems were found indistinguishable from human-written ones.

3.3 Constrained Text Generation

Constraints on text generation have been studied not only for creative purposes but also for various other aspects. For instance, ensuring that a translation is formal enough or that the generated text contains a specific list of words. [26] The upcoming section will cover different techniques for adding constraints to text generation with auto-regressive transformer models, which are not limited to generating parodies.

3.3.1 Adding Control Codes

To be able to have more control of text generation, Keskar et al. introduced CRTL [15], a publicly available language model that is very large and powerful. It is trained using 55 different control codes, which enable the user to provide more specific information about the text that they want to generate. For example, the user can define the domain, entities, relationships, and other task-specific information. Control codes guide the model to focus on parts of the model that contain the relevant information to generate the desired text.

3.3.2 Plug and Play Language Models

Dathari et al. introduced a new way of guiding an auto-regressive pre-trained large language model without fine-tuning it. This method is called Plug and Play Language Models (PPLM) [9]. The PPLM framework combines an unconditioned pre-trained language model, such as GPT-2, with one or more attribute models that represent the desired constraints or goals. These attribute models can be smaller than the base language model and can be simple classifiers or bag-of-words models. In the PPLM architecture, at each step of text generation, the pre-trained language model generates a history matrix H_t that encapsulates the context of the generated sequence up to that point. This matrix is then modified using gradients derived from the attribute models and the base model by a forward and backward pass. These gradients are applied to H_t to adjust the LM's activations in the direction that increases the likelihood of generating text that aligns with the specified attributes and aligns with the original distribution to ensure fluent text. This method also adds a level of controllability, allowing for influence over the effect of the updates. As this optimization process is done ex post facto, there is no need to retrain the original model. Through various controlled text generation tasks, PPLM has demonstrated its effectiveness in steering a model like GPT-2 toward specific topics, styles, and sentiments, outperforming fine-tuned versions in producing attribute-aligned and fluent text.

3.3.3 GeDi

Krause et al. introduced a new method called GeDi (Generative Discriminator Guided Sequence Generation) [16] for controlling the text generation process of large language models. GeDi utilizes smaller class-conditional language models (CC-LMs) as generative discriminators, that steer the primary LM towards generating text that aligns with specific attributes or constraints. To achieve this, the classification probabilities for each potential next token are computed by contrasting the predictions of the CC-LMs, using Bayes's rule for partial sequences. The generative discriminators modify the token probabilities of the base LM at each generation step, by adjusting the probabilities with a Bayes decomposition, in favor of the tokens that align with the attribute or constraint while disfavoring those aligned with the undesired constraint. Similar to nucleus sampling, a certain amount of the top-tokens is retained, so that the cumulative probability is larger than a set ρ. GeDi outperforms state-of-the-art methods when tested on different controlling tasks, and it shows a significant increase in speed.

3.3.4 Training Experts

Liu et al. devised a method called DExperts [20] to guide text generation toward a particular constraint without having to retrain the entire model. The method uses two additional experts, namely an expert and an anti-expert, which are pre-trained language models that can be smaller in size than the base model. The expert is fine-tuned to generate text that adheres to the desired constraint or behavior, while the anti-expert is trained to generate text that does the opposite. To incorporate the experts into the base model, after each pass the logits from the expert and the anti-expert are multiplied with a hyperparameter α and are then added and subtracted to the generated logits of the base model, respectively. This is done before applying the softmax function. The hyperparameter is included to tune the involvement of the experts. Only the tokens that are part of the top-k/top-p for the base model are considered for generating text, while all other tokens are assigned a value of negative infinity. The effectiveness of this method was tested on two tasks: toxicity avoidance and sentiment-controlled generation. The tests were conducted using GPT-2 Large as the base model, and different sizes of GPT-2 for the experts and anti-experts. The results showed that the system was able to guide text generation toward the desired constraint while producing diverse and fluent text.

3.3.5 FUDGE

Yang and Klein [56] introduce FUDGE or Future Discriminators for Generation, a flexible and modular approach for controlled text generation that uses a pre-existing language model (G) without altering or fine-tuning it. FUDGE operates by learning a discriminator per constraint, which predicts how probable it is that the constraint will be met in the future given a sequence. This approach follows a Bayesian factorization, where the probability of a token given the sequence and the constraint is proportional to the product of the base model's probability for the token and the discriminator's probability of the constraint given the sequence with the token. This allows for the integration of constraints into the generation process without altering the underlying language model. The discriminators in FUDGE are designed to be attribute specific, allowing for a high degree of modularity. This means that for different attributes or constraints that one might want to impose on the text generation, separate discriminators can be trained and then applied as needed using Bayesian decomposition. The only downside to this method is that it cannot guarantee that constraints will be met. The FUDGE method was tested in three different applications. First, in poetry couplet completion, FUDGE showed its ability to generate text that met specific poetic constraints while maintaining linguistic coherence and creativity. Second, in topic-controlled language generation, the method successfully guided the language model to generate text that was not only topical, but also coherent and diverse, showcasing its ability to control content without stifling creativity or variety in expression. Finally, in formality change in machine translation, the model adeptly modified the formality of translated text while preserving the original meaning.

3.3.6 Multi-Objective Controlled Optimization

Kumar et al. propose MuCoCO (Multi-Objective Controlled Optimization) [17], another approach to control text generation by incorporating constraints when using pre-trained language models. Similarly to PPLM they use gradients from the attribute models that are trained on the specific constraint, to guide the base model. However, instead of altering the model activations in an autoregressive manner, the gradients are used to influence the probabilities iteratively to adhere to multiple constraints simultaneously. MuCoCO formulates the decoding process as a multi-objective optimization problem, where each constraint is represented as a differentiable function. It achieves this by converting the discrete optimization task of text generation into a continuous one. Each token in the generated sequence is represented as a probability simplex over the vocabulary of the base model, creating a soft representation of the sequence, which is initialized uniformly. It is seen as a constraint optimization problem and by using gradient descent and Lagrange multipliers the soft sequence is iteratively adjusted to satisfy the constraints. The use of gradient descent ensures that the generated sequence converges towards a solution that satisfies the constraints while retaining the base model's original distribution. MuCoCO was tested on multiple combinations of constraints, style transfer, and style-controlled machine translation and showed effective results.

3.3.7 NRETM

Wang et al. introduce a new method, NRETM (Neural Rule-Execution Tracking Machine) [54], to guide pre-trained transformer-based text generators to adhere to certain constraints. It does this by feeding the current state of all of the constraints back to the language model as basically an extra input. All constraints are expressed in predicate logic, and for each predicate, there is an executable program that determines the state of the constraint. During each generation step the Logic Tracker calculates the state of each constraint and stores the states in the State Matrix. A transformer-based encoder called State Matrix Encoder takes the State Matrix as input and provides its information to the base model. Both the State Matrix Encoder and the base model need (re)training. NRETM shows promising results when tested across different text generation tasks.

3.3.8 Controlled Decoding

Mudgal et al. propose Controlled Decoding [28], a new scoring method to guide a pre-trained base text generation model towards a certain goal. Similarly to FUDGE, it trains an extra transformer-based model that adjusts the probabilities of possible tokens while generating each token. The constraints are given as a reward function that can be given to complete sequences, e.g. one that has the EOS token. To apply such a reward on token level, Mudgar et al. present an optimal solution to adapt the probabilities of the original base model. This solution consists of a value function, which gives the reward if the next token is the EOS token or a prediction of the potential reward otherwise, which can be derived from the optimal solution. To approximate this value function or prefix scorer, a transformer-based model is trained with a given reward function for one or more constraints. They also present a new sampling method, blockwise sampling, where the base model produces k independent sequences of length m and picks the one with the best prefix score to build on. Their methods proved their effectiveness when tested on dialog length and safety. Interestingly, the blockwise sampling method had a much better trade of curve when the method effectiveness was compared to KL divergence from the original language model distribution.

3.3.9 NADO

Meng et al. [26] introduce a general framework to control generation with sequence-level constraints on auto-regressive transformer models. They achieve this by using a pre-trained base model and a Neurally-Decomposed Oracle. The constraints are on sequence-level, and must be expressed as a Boolean function, meaning that given a sequence, the function returns True if it adheres to the constraint and false otherwise. To constrain the generation on token-level, Meng et al. derive a closed-form optimal solution to turn the sequence-level oracle function into a token-level function that guides the transformer model to adhere to the constraint. However, since the solution is intractable, an approximation is used. The approximation called the *NeurAlly-Decomposed Oracle* (NADO), is a Seq2seq model trained with samples from the base model to match its distribution better. During inference, both the base model and NADO run in parallel, and their distributed outputs are added together element-wise. The framework was tested under two different constraints: lexical constraints and enforcing formality when translating. Both constraints yielded promising results and scored better than state-of-the-art solutions.

3.3.10 Tractable Control: GeLaTo

Zhang et al. [58] introduce GeLaTo (Generating Language with Tractable Constraints), a framework that applies tractable probabilistic models (TPMs) to impose lexical constraints on autoregressive language generation, particularly in large models like GPT-3, without retraining the base model. The TPM is a model that simulates the same distribution as the base language model but can easily be used to integrate specific lexical constraints into the generation process. GeLaTo uses a two-step approach to achieve this. First, the TPM is trained via Maximum Likelihood Estimation on samples from the base language model, effectively reducing the Kullback-Leibler (KL) divergence between the TPM and the base LM. Secondly, during the generation process, GeLaTo computes the next-token probability conditioned on the lexical constraints, combining this with the base model's probabilities to generate fluent text that adheres to the specified constraints. The Markov model is trained only on samples from the base model, and constraints are applied only during inference. As a result, there is no need to retrain the auxiliary model to add constraints. This is in contrast to FUDGE or NADO, which require the retraining of their auxiliary model for any new constraints. GeLaTo demonstrates remarkable results on challenging constrained text generation datasets, such as CommonGen, News, and Yelp! Review. It achieves state-of-the-art quality while ensuring 100% constraint satisfaction.

3.3.11 InstructCTG

Zhou et al. propose a novel framework, InstrcutCTG [60], to control constraints for text generation with pre-trained language models. This is achieved solely through fine-tuning the base model, similar to CRTL. However, instead of using control words, the constraint is verbalized in the prompt using a certain fixed format per constraint. Their model is fine-tuned on five different constraint types: lexical, syntactic, semantic, style, and length constraints. In comparison to other state-of-the-art methods, it has demonstrated similar or better quality with much faster response times, as the decoding process remains unaltered.

3.3.12 Regular Expression Instruction

Zheng et al. [59] improved on InstructCTG by introducing Regular Expression Instructions to control text generation with certain constraints. Instead of expressing the constraints in natural language, they make use of markup language to indicate the constraints more precisely. They only fine-tune language models with the constraints embedded with markup language in the prompt. If the generated output does not satisfy the constraints defined in the prompt, they reject the outputs and retry at most k times. This approach shows promising results and outperforms most of the state-of-the-art models against which it was compared.

3.3.13 Constrained Beam Search for Image Captioning

Anderson et al. [2] introduce a new decoding strategy for text generation, Constrained Beam Search. This strategy helps to enhance the capabilities of image captioning models, especially for images containing new or out-of-domain content. This method addresses a limitation in previous image captioning models, which typically struggle to caption images that feature scenes or objects not covered in their training datasets. However, many images already come with image tags that can help improve the captioning of those images. Anderson et al. found a way to enforce those tags into the generated caption. This enforcement is achieved by constraining the beam search when decoding the output, with a finite-state machine (FSMs). The FSM represents the constraints that need to be enforced and in which way they can be achieved. The FSM will have a starting state, where none of the constraints are met and one or more ending states where all the constraints are met. The constrained beam search algorithm works by maintaining multiple beams, one for each state of the FSM. At each step of the decoding process, the algorithm considers all possible extensions of current sequences in each beam. It then selects the most probable sequences that also comply with the state transitions dictated by the FSM. This method allows the model to satisfy the constraints imposed by the image tags. The generation stops when a hypothesis ends with an end-of-sequence tag and is in an ending state of the FSM. By guiding the generation of captions with image tags and ensuring the inclusion of these tags, this approach significantly improves the quality and relevance of captions for images with new or out-of-domain content.

3.3.14 Grid Beam Search

Hokamp and Liu present a new method called Grid Beam Search (GBS) [12], which is an adaptation of beam search that incorporates lexical constraints into sequence generation models. This method enables the addition of specific information without modifying or retraining the original generation model. The GBS algorithm restructures the decoding process into a grid system, that is indexed by time steps and the number of constraints satisfied. Each grid slot captures the best k-hypotheses. A hypothesis can either be classified as 'open', which allows tokens to be generated from the model's distribution or initiates a constraint, or 'closed', which restricts the generation to the continuation of the current constraint sequence. During each step of decoding, GBS generates new hypotheses through three primary operations: generation from the model's distribution for open hypotheses in the current beam, initiation of new constraints by open hypotheses from the preceding beam, and continuation of constraints by closed hypotheses. This method ensures that all potential paths that satisfy the constraints are explored without altering the core mechanism of the model. The end-of-sequence token can only be generated in the top row of the grid where all the sequences have met the constraints. The best-finished sequence from that row is chosen as the final output, ensuring the output meets all the constraints. The algorithm shows promising results in various applications, such as interactive machine translation and domain adaptation for Neural Machine Translation. Its ability to adaptively integrate user inputs or domain-specific terminologies as constraints significantly enhances the model's use in producing accurate outputs.

3.3.15 Dynamic Beam Allocation

Post and Vilar introduce Dynamic Beam Allocation (DBA) [39], a successor to GBS [12], discussed earlier. DBA is an algorithm that efficiently manages lexical constraints within beam search for neural machine translation. It is a significant improvement over GBS as the beam size stays constant, unlike GBS, where the beam size scales linearly with the number of constraints. The DBA algorithm divides the beam search process into several distinct "banks," each of which corresponds to hypotheses that have satisfied an equal number of lexical constraints. For example, one bank might contain hypotheses that have met two constraints, another bank might contain hypotheses that have met three constraints, and so on. At each decoding step, the system dynamically allocates the fixed-size beam evenly across these banks. The candidate selection process in DBA is more intricate compared to standard beam search. It not only selects the top-k tokens overall but also includes tokens that extend unmet constraints and the single best token from each hypothesis. Afterward, those candidates are placed in their corresponding bank. All this ensures that the constraints are progressively met. The end-of-sequence token, to indicate a finished hypothesis, can only be generated on hypotheses that have met all the constraints. When tested with an RNN architecture, the algorithm confirms the speed gains compared to GBS, even with an increase in constraints.

Hu et al. [14] have updated the DBA algorithm with three notable improvements. To address the scenarios where two sequences share a common subsequence, they introduce a compact multi-state trie to represent the constraints. This allows for overlapping constraints and keeps track of the count of each constraint in a hypothesis. Every hypothesis has its trie where each end node has a counter to indicate how many times that constraint must be generated. The counter is decremented every time the constraint appears in the sequence. The second addition to DBA is the ability to handle negative constraints by setting the logits of forbidden tokens to minus infinity. Finally, the algorithm has been converted to a vectorized one for parallelizing the search of new candidates, using GPU operations. The modified DBA algorithm was tested on multiple constrained decoding tasks and showed enhanced speed compared to the original one.

3.3.16 RESEAL

Chen et al. introduce a new method RESEAL (RElation-guided probability Surgery and bEam ALlocation) [8], that enforces both lexical and relational constraints in text generation. Relational constraints are specific requirements that dictate the relationship between words in a generated sequence. RESEAL builds upon the Dynamic Beam Allocation (DBA) framework but enhances it by including a relation identifier. This identifier helps adjust the probability distributions for candidate words during the generation process, ensuring that predefined relational constraints are met. The method operates in two stages. Firstly, in the *Probability Surgery* phase, RESEAL modifies the probability distribution for each candidate word based on the relation constraints, to maintain the desired relations. Secondly, RESEAL uses an adaptation of the DBA algorithm to enforce both the lexical and relational constraints. To also take into account the relations constraints, the banks are not divided based on the number of lexical constraints met, but on the number of correct relational constraints met. When tested on dependency placement and downstream tasks, such as sentence summarization, fact-based text editing, and data-to-text generation, RESEAL shows very promising results in meeting both lexical and relational constraints, while still retaining a fluent and coherent text.

3.3.17 Directed Beam Search

Pascual et al. introduce a new method called Directed Beam Search (DBS) [34] to guide text generation to follow certain constraints. DBS is mainly focused on lexical constraints, where specific keywords must be included in the generated text, while still maintaining coherence and fluency. It is a plug-and-play approach that can be used with any forward-generating language model, such as auto-regressive transformer models, without the need for fine-tuning or retraining, making it energy-efficient.

DBS guides the generation to follow the lexical constraints in two steps. Firstly, after each pass of the language model, it changes the received logits to favor words similar to the keywords, before applying the usual softmax on the logits. Secondly, a modified version of beam search is used. For each of the b beams k tokens are generated. Then, each beam is extended s times with k tokens. After that, a quality score is calculated for each of the b x s sequences, based on how many of the required words it has, repetition of words, and its perplexity. Finally, the b beams with the best score are kept and the cycle continues until generation stops. Tests with GPT-2 [42] show promising results, with a 93% accuracy on adhering to the constraint and lower perplexity than GPT-2 without beam search.

3.3.18 Neurologic Decoding

Lu et al. present NeuroLogic Decoding [24], a new beam-search decoding method to lexically constrain text generation with predicate logic constraints and pre-trained models. This method accepts all lexical constraints that can be expressed under predicate logic with Conjunctive Normal Form. Every disjunction is treated as one clause. The novelty of this method lies in how the k-candidates for the beam are selected. The algorithm consists of three steps: pruning, grouping, and selecting. In the pruning step, all candidates that have clauses that can not be satisfied anymore are discarded. Additionally, the candidates that do not fall in the top-α (ranked on their likelihood) and the top-β (ranked on the number of satisfied clauses) are also discarded. In the grouping step, the remaining candidates that have the same constraints satisfied are grouped, and the beam is then filled with the best ones of each group. They are selected based on their score, determined by their likelihood and whether they partially satisfy an unsatisfied constraint. When all scores are calculated, the different groups are ranked according to the best score of each group.

The beam is then filled by taking one from each group in rotation until it is filled. The method was tested on four different tasks, and the results show that it outperforms other decoding strategies, both in accuracy and speed.

3.3.19 Neurlogic A*esque Decoding

Lu et al. [23] improved upon Neurologic Decoding by adding A*-like Decoding to consider future heuristics. This allows them to select the path that is most likely to satisfy all constraints while still producing coherent and fluent text. To achieve this, they added a future cost to the score of the sequences which is proportional to the likelihood of satisfying an unsatisfied constraint in the next l-tokens, starting from the newly added token. They propose three ways to look into the future: greedy search, sampling, and beam search. The first two generate a single sentence with their respective search method, while beam search returns the k-best sequences. To calculate the likelihood of a constraint being fulfilled, they take the maximum probability that the constraint will be generated somewhere along the future sequence or sequences. This method was tested on four different constrained tasks and it outperformed all prior methods on all four tasks. It had almost perfect constraint satisfaction and showed the most improvement in complex constraints with multiple terms. Additionally, they tested their algorithm on unconstrained generation by considering the probabilities of the lookahead sequences, when selecting the next token. This improved fluency and coherence but also gave more interesting stories.

3.3.20 COLD Decoding

Qin et al propose COLD Decoding (Energy-based Constrained Decoding with Langevin Dynamics) [41], a novel method to constrain text generation without fine-tuning or altering the pre-trained base model. This method utilizes an energy-based model, where each constraint is defined as a continuous function. These functions return an energy score given a vector sequence, with higher scores indicating better adherence to the constraints. The method first represents the text sequence in the continuous space as a soft sequence consisting of vectors of the size of the vocabulary of the language model for each time stamp. These vectors can be seen as the logit output, similar to the output of the language model. The algorithm then uses Langevin Dynamics to iteratively refine the vectors of the sequence using the gradient of the total energy function, which contains all constraint functions. To ensure fluency, the energy function often includes a constraint function to mimic the original language model. After N iterations, the algorithm produces a sequence of vectors representing the logits for each time stamp. However, because using them directly would result in incoherent and illogical text, the language model is used to generate the text. At each generation step, the best k-tokens are sampled according to the language model and modified according to their score given by the vector at the corresponding time stamp. COLD was tested on various constraining tasks, such as abductive reasoning, counterfactual writing, and lexical constrained writing. The results show its effectiveness and its ability to create fluent and coherent text while still adhering to given constraints. Furthermore, it performs similarly to other state-of-the-art methods like Neurologic.

3.3.21 NeuroStructural Decoding

Bastan et al. introduce NeuroStructural Decoding [4], an improvement of Neuro-Logic Decoding that also includes semantic constraints. It focuses on three sorts: *unary*,*binary*, and *triplet* to enforce a syntactic position, a single dependency, and a two-way relation respectively. To validate the semantic relations of a sentence a dependency parser is required. However, these parsers are generally trained on full sentences, and when generating partial sentences, they need to establish relations accurately as well. Therefore the authors retrain a dependency parser from the CoreNLP [25] library, with partial sentences to achieve greater accuracy. Similar to NeuroLogic, the method generates for each beam n possible candidates. Each candidate is then parsed for its dependencies and those that irreversibly violate any of the constraints are pruned away. The possible hypotheses are then grouped based on the number of correct clauses, not lexical constraints. The score is based on the hypothesis's original probability and the penalties for the number of clauses they have not met yet. The selection process within each group and the process to fill the final beam is unchanged. The method was tested on three different tasks and showed improvements compared to vanilla and NeuroLogic decoding.

3.4 Backtracking & Search Algorithms

[TODO]

Chapter 4

Problem Statement

[TODO]

Chapter 5

Approach

[TODO]

Chapter 6

Evaluation

[TODO]

Chapter 7

Conclusion

The final chapter contains the overall conclusion. It also contains suggestions for future work and industrial applications. [TODO]

Appendices

#Appendix A

Examples of Chatbots Writing Haiku

The following are examples of haiku written by various chatbots. They were all given the same simple prompt: *"Write a haiku about how crazy pineapple on a pizza is"*. These examples were generated to demonstrate that chatbots and thus language models do not naturally follow the rules associated with certain text formats, such as poems. Specifically, the following examples are about writing haiku, which are poems consisting of three lines with a 5-7-5 syllable structure. The chatbots used in these examples have a nondeterministic character, which can be seen by giving the chatbot the same prompt twice, which will result in different results. All the syllable counts mentioned were obtained with https://syllablecounter.net/count. Some of them have to correct structure or are very close to it. Some of the haiku have the correct structure or are very close to it. This shows that language models already know what a haiku contains but do not always adhere to the associated constraints.

A.1 ChatGPT 3.5 Turbo

The two examples below are obtained from the ChatGPT 3.5 Turbo model[31] from OpenAI. The first one is also presented in the introduction of this thesis to show its 5-6-5 syllable structure. When the chatbot was asked a second time, it generated a different haiku that this time adhered to the 5-7-5 syllable structure. This shows its nondeterminism and that it does have some notion about the 5-7-5 rule, but does not always know how to exactly apply it. The following examples were both generated on 12/12/2023.

Example 1

Golden waves of heat
Pineapple dances wild
Pizza's sweet embraced

Example 2

Golden sun on dough,
Crazy dance of pineapple
Pizza dreams take flight

A.2 Bard

The same prompt was also tested on Bard[11] which is supported by the Gemini model [50]. It generated the poem below. Compared to ChatGPT the example deviates much more from the 5-7-5 syllable structure with a 6-9-6 one. It shows that the model has some notion about what a haiku is. It gives a three-line poem with a similar structure, but it has difficulty with the exact syllable count. The example was generated on 12/12/2023.

Example

Sweet, tangy fruit's delight
On savory dough, a clash so grand
A culinary crime

A.3 Llama 2

The 70B chat model of Llama 2 [52] was also tested for its haiku abilities[1]. The model generated a haiku which is almost in line with the 5-7-5 syllable structure; it has a 5-7-4 structure according to the used syllable counter. When counting *'doesn't'* as two syllables it can be seen as a correct haiku. This example was generated on 25/01/2024.

Example

Crazy pineapple
On a pizza, oh so wrong
Fruit doesn't belong

[1]It was tested on the site Llama2.ai, which hosts the vanilla versions of the LLama 2 chat models

Bibliography

[1] Grammarly. https://www.grammarly.com.

[2] P. Anderson, B. Fernando, M. Johnson, and S. Gould. Guided open vocabulary image captioning with constrained beam search. In M. Palmer, R. Hwa, and S. Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945. Association for Computational Linguistics.

[3] D. Andersson. AI generated parody lyrics, fall 2021.

[4] M. Bastan, M. Surdeanu, and N. Balasubramanian. NEUROSTRUCTURAL DECODING: Neural text generation with structural constraints. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9496–9510. Association for Computational Linguistics.

[5] J. Belouadi and S. Eger. ByGPT5: End-to-end style-conditioned poetry generation with token-free language models. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7364–7381. Association for Computational Linguistics.

[6] T. Chakrabarty, V. Padmakumar, and H. He. Help me write a poem: Instruction tuning as a vehicle for collaborative poetry writing. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6848–6863, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[7] J.-W. Chang, J. C. Hung, and K.-C. Lin. Singability-enhanced lyric generator with music style transfer, 2021.

[8] X. Chen, Z. Yang, and X. Wan. Relation-constrained decoding for text generation.

[9] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models: A simple approach to controlled text generation.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. Number arXiv:1810.04805. arXiv, 2019.

[11] Google. Google bard, 2023. Accessed: January 25, 2024.

[12] C. Hokamp and Q. Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546. Association for Computational Linguistics.

[13] J. Hopkins and D. Kiela. Automatically generating rhythmic verse with neural networks. In R. Barzilay and M.-Y. Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 168–178, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[14] J. E. Hu, H. Khayrallah, R. Culkin, P. Xia, T. Chen, M. Post, and B. Van Durme. Improved lexically constrained decoding for translation and monolingual rewriting. In *Proceedings of the 2019 Conference of the North*, pages 839–850. Association for Computational Linguistics.

[15] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. CTRL: A conditional transformer language model for controllable generation.

[16] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, and N. F. Rajani. GeDi: Generative discriminator guided sequence generation.

[17] S. Kumar, E. Malmi, A. Severyn, and Y. Tsvetkov. Controlled text generation as continuous optimization with multiple constraints.

[18] W. Lau. Application of machine learning model in generating song lyrics. 2021.

[19] P. Li, H. Zhang, X. Liu, and S. Shi. Rigid formats controlled text generation. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 742–751. Association for Computational Linguistics.

[20] A. Liu, M. Sap, X. Lu, S. Swayamdipta, C. Bhagavatula, N. A. Smith, and Y. Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts.

[21] N. Liu, W. Han, G. Liu, D. Peng, R. Zhang, X. Wang, and H. Ruan. ChipSong: A controllable lyric generation system for chinese popular song. In *Proceedings of the First Workshop on Intelligent and Interactive Writing Assistants (In2Writing 2022)*, pages 85–95. Association for Computational Linguistics, 2022.

[22] K.-L. Lo, R. Ariss, and P. Kurz. Gpoet-2: A gpt-2 based poem generator, 2022.

[23] X. Lu, S. Welleck, P. West, L. Jiang, J. Kasai, D. Khashabi, R. Le Bras, L. Qin, Y. Yu, R. Zellers, N. Smith, and Y. Choi. NeuroLogic a*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799. Association for Computational Linguistics.

[24] X. Lu, P. West, R. Zellers, R. L. Bras, C. Bhagavatula, and Y. Choi. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints.

[25] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[26] T. Meng, S. Lu, N. Peng, and K.-W. Chang. Controllable text generation with neurally-decomposed oracle.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. 2013.

[28] S. Mudgal, J. Lee, H. Ganapathy, Y. Li, T. Wang, Y. Huang, Z. Chen, H.-T. Cheng, M. Collins, T. Strohman, J. Chen, A. Beutel, and A. Beirami. Controlled decoding from language models. Publisher: arXiv Version Number: 1.

[29] H. G. Oliveira. WeirdAnalogyMatic: Experimenting with analogy for lyrics transformation, 2020.

[30] H. G. Oliveira. Exploring a masked language model for creative text transformation. 2021.

[31] OpenAI. Chatgpt-3.5 turbo. `https://www.chat.openai.com`, 2022.

[32] OpenAI. Chatgpt-4. `https://www.chat.openai.com`, 2023.

[33] A. Ormazabal, M. Artetxe, M. Agirrezabal, A. Soroa, and E. Agirre. PoeLM: A meter- and rhyme-controllable language model for unsupervised poetry generation. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3655–3670, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[34] D. Pascual, B. Egressy, F. Bolli, and R. Wattenhofer. Directed beam search: Plug-and-play lexically constrained language generation.

[35] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[36] S. Pichai and D. Hassabis. Google gemini: Next-generation model. https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#sundar-note, 2024. Accessed: 2024-02-25.

[37] A. Popescu-Belis, À. Atrio, V. Minder, A. Xanthos, G. Luthier, S. Mattei, and A. Rodriguez. Constrained language models for interactive poem generation. In N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, J. Odijk, and S. Piperidis, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3519–3529, Marseille, France, June 2022. European Language Resources Association.

[38] A. Popescu-Belis, À. R. Atrio, B. Bernath, E. Boisson, T. Ferrari, X. Theimer-Lienhard, and G. Vernikos. GPoeT: a language model trained for rhyme generation on synthetic data. In S. Degaetano-Ortlieb, A. Kazantseva, N. Reiter, and S. Szpakowicz, editors, *Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 10–20, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[39] M. Post and D. Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324. Association for Computational Linguistics.

[40] P. Potash, A. Romanov, and A. Rumshisky. GhostWriter: Using an LSTM for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924. Association for Computational Linguistics, 2015.

[41] L. Qin, S. Welleck, D. Khashabi, and Y. Choi. COLD decoding: Energy-based constrained text generation with langevin dynamics.

[42] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2018.

[43] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[44] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[45] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Number arXiv:1910.10683. arXiv, 2020.

[46] N. Ram, T. Gummadi, R. Bhethanabotla, R. J. Savery, and G. Weinberg. Say what? collaborative pop lyric generation using multitask transfer learning. In *Proceedings of the 9th International Conference on Human-Agent Interaction*, HAI '21, pages 165–173. Association for Computing Machinery, 2021.

[47] M. Riedl. Weird AI yankovic: Generating parody lyrics. Number arXiv:2009.12240. arXiv, 2020.

[48] M. Riedl. weirdai. https://github.com/markriedl/weirdai, 2020.

[49] M. A. Rodrigues, A. Oliveira, A. Moreira, and M. Possi. Lyrics generation supported by pre-trained models, 2022.

[50] G. Team et al. Gemini: A family of highly capable multimodal models.

[51] Y. Tian and N. Peng. Zero-shot sonnet generation with discourse-level planning and aesthetics features. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3587–3597, Seattle, United States, July 2022. Association for Computational Linguistics.

[52] H. Touvron et al. Llama 2: Open foundation and fine-tuned chat models.

[53] T. Van de Cruys. Automatic poetry generation from prosaic text. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2471–2480, Online, July 2020. Association for Computational Linguistics.

[54] Y. Wang, C. Xu, H. Hu, C. Tao, S. Wan, M. Dras, M. Johnson, and D. Jiang. Neural rule-execution tracking machine for transformer-based text generation.

[55] L. Xue, K. Song, D. Wu, X. Tan, N. L. Zhang, T. Qin, W.-Q. Zhang, and T.-Y. Liu. DeepRapper: Neural rap generation with rhyme and rhythm modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 69–81. Association for Computational Linguistics, 2021.

[56] K. Yang and D. Klein. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535.

[57] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. XLNet: Generalized autoregressive pretraining for language understanding, 2020.

[58] H. Zhang, M. Dang, N. Peng, and G. V. d. Broeck. Tractable control for autoregressive language generation.

[59] X. Zheng, H. Lin, X. Han, and L. Sun. Toward unified controllable text generation via regular expression instruction. Publisher: arXiv Version Number: 2.

[60] W. Zhou, Y. E. Jiang, E. Wilcox, R. Cotterell, and M. Sachan. Controlled text generation with natural language instructions. Publisher: arXiv Version Number: 2.