University of Nottingham
UK | CHINA | MALAYSIA

Machine Learning in Science
Department of Physics and Astronomy

Optimising Neural Text Generation via Biased Trajectory Sampling

Author Jack Paine

Supervisors Jamie Mair
Edward Gillman
Juan Garrahan

A dissertation submitted in partial
fulfilment for the degree of
Master of Science

September 2024

Abstract

Large language models (LLMs) have shown unprecedented capabilities in a multitude of natural language processing (NLP) tasks, including text and code generation. The text generation process can be conceptualised as a series of decisions; an input text is provided, and the LLM repeatedly chooses a 'token' (character, word, or punctuation) to append to the text. It is common to inject random noise into this decision-making process via the use of a *decoding strategy*, helping to make the generated text appear more like human-written text. However, this typically comes at the expense of accuracy, as the chance of an incorrect token getting sampled along the sequence is increased. We propose a novel decoding strategy, using a transition path sampling (TPS) algorithm from molecular dynamics to effectively search the output space of potential sequences. Our method offers finer control over the exploration-exploitation trade-off in text generation compared to existing strategies, showing an increase in both accuracy and diversity of generated text. This is backed by tests on the HumanEval code generation benchmark, for which our strategy beats both greedy search and beam search. This points to the untapped potential of unconventional sampling strategies to improve the quality of LLM-written text.

Acknowledgements

I'd like to give a huge thank you to my project supervisors, Jamie Mair and Edward Gillman, whose guidance and knowledge have been invaluable throughout this project.

I am also grateful for access to the University of Nottingham's ADA High Performance Computing service, which has been instrumental in the research and development of this project.

Finally, thank you to HuggingFace for hosting the models used in this project.

Contents

List of Acronyms

LLM large language model

NLP natural language processing

SOTA state-of-the-art

PMF probability mass function

CLM causal language model

MLM masked language model

EOS end-of-sequence

PMF probability mass function

MC Monte Carlo

MCMC Markov chain Monte Carlo

TPS transition path sampling

OTG open text generation

Definitions

token: a single unit of text, which may be a word, part of a word, or some punctuation
sequence or **trajectory**: usually refers to a sequence of tokens
prompt: the input (text or tokens) given to a model to generate a completion
completion: the output (text or tokens) of a neural text generation process
decoding strategy or **strategy**: a method of generating text with a language model
local strategy: outputs a single token at each step
global strategy: considers the search space of full completions
quality: the joint probability of a sequence appearing under the model's unaltered probability distribution

Notation

$V = \{0, 1, 2, \ldots, n-1\}$: model vocabulary of n tokens
$X = [x_0, x_1, x_2, \ldots, x_t]$: sequence of input tokens, $x_i \in V$, length t
$C = [c_0, c_1, c_2, \ldots, c_m]$: sequence of completion tokens, $c_i \in V$, length m
$Y = [y_0, y_1, y_2, \ldots, y_{n-1}]$: model output logits corresponding to each token in V
$P(v) = P(v \mid X)$: probability that next token in sequence X is $v \in V$
$P(V)$: full probability distribution over vocabulary V
$\tilde{P}(V)$: transformed probability distribution over vocabulary V
$Q(C \mid X) = \frac{1}{m} \log P(C \mid X)$: quality of completion C with length m, given input X
$x_{a:b}$: subsequence of tokens from positions a to b inclusive
$[A; B]$: concatenation of sequences (or tokens) A and B, e.g. $[a_0, a_1, a_2, b_0, b_1]$

1 Introduction

Large language models (LLMs, or simply 'language models') are deep neural architectures trained on vast text corpora to generate human-like text. Stemming from the field of natural language processing (NLP), LLMs have shown unprecedented capabilities across a wide domain of machine-language tasks, with performance that rivals – or even exceeds – specialist systems in translation, summarisation, creative writing, question answering, and code generation tasks [1, 2]. Some speculate that the current state-of-the-art (SOTA) language models show indications of generalised intelligence [3], and randomised trials have shown that humans cannot reliably distinguish between human and LLM-generated text [4].

However, LLMs only form one component of the text generation process; the choice of *decoding strategy* can also largely impact the output. For a given input text, the LLM assigns a score $P(v)$ to each *token* from its vocabulary – which may be a word, part of a word, or some punctuation – corresponding to the probability that it appears next in the sequence. It is then the role of the decoding strategy to 'translate' this probability distribution P into the actual continuation of the sequence, Figure 1.1. This is repeated to continuously generate text until some stopping criterion (such as sequence length) is reached.

This report seeks to explore the impact of different decoding strategies on the quality and diversity of the generated text and to propose a novel decoding strategy that balances these two factors.

1.1 Problem Statement

Text generation can be viewed as a series of decisions, with the decoding strategy choosing the most suitable token to add to the sequence at each step. A basic approach, known as 'greedy search' decoding, involves selecting the highest-scoring token at each step. However, this often results in repetitive and uninteresting text completions [5], as illustrated in Figure 1.2a. While

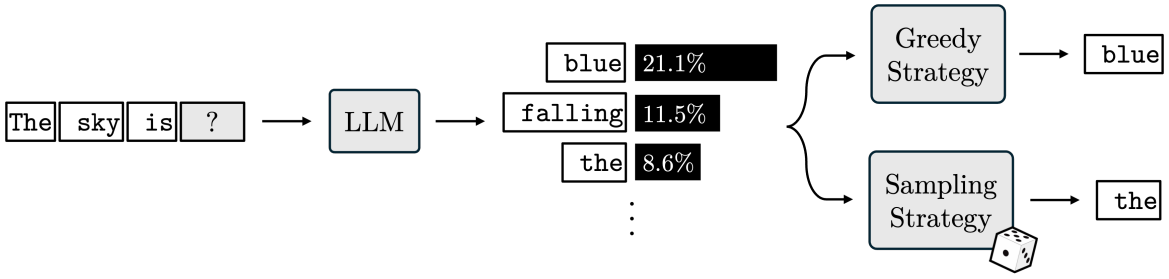


Figure 1.1: The role of decoding strategies in text generation. The model produces scores corresponding to each potential sequence continuation, which the decoding strategy uses to select a token.

Starting text: "`The capital of France is`"

(a) Greedy search decoding strategy

```
The capital of France is Paris.
The capital of Germany is Berlin.
The capital of Greece is Athens.
The capital of Italy is Rome.
The capital of Japan is Tokyo.
```

(b) Random sampling decoding strategy

```
The capital of France is one of
the most beautiful cities in the
world. Families packed the Eiffel
Tower itself, again to ignite the
imagination for its senses. You
```

Figure 1.2: Example showcasing different decoding strategies to generate text. Note how (a) seems repetitive and predictable, while (b) is more creative but lacks coherence. *Generated using Llama 2 7B* [7]

the probabilities provided by the model may be accurate, choosing the most likely token at each step cannot reflect the diversity and randomness seen in human-written text [6]. It is instead more common to use a stochastic decoding strategy, such as *random sampling*[1], to reduce repetition and predictability in the text. However, introducing randomness naturally makes irrelevant tokens more likely to be selected at each step, periodically leading to incoherent responses [6].

The trade-off between accuracy and response diversity poses a significant challenge in neural text generation, as existing strategies often struggle to maintain coherence while introducing diversity. Generating human-like responses is important for enabling natural human-computer interaction, but it is equally important that the text is accurate and coherent. Various solutions to this issue have been proposed, such as *temperature sampling* and *top-p sampling* [6], which aim to promote sequence diversity without mitigating the integrity of the generated text. However, these approaches do not necessarily consider the overall context and consistency of the output and thus may still lead to quality degradation. Global pathfinding strategies, such as beam search and best-of-n, can be utilised make more informed decisions by considering the entire sequence when generating text; this typically comes at the expense of computational efficiency and reduced diversity.

1.2 Research Aim and Objectives

In this project we explore the use of Markov Chain Monte Carlo (MCMC) and transition path sampling (TPS) to bias the distribution of machine-generated text. Specifically, we employ a 'forward shooting' algorithm [8], which borrows from the field of molecular dynamics to generate sequences that are both coherent and diverse. We evaluate the performance of this biased decoding strategy against a set of benchmarks covering both code generation and general text generation tasks, and compare it against other decoding strategies, including greedy search, random sampling, and global pathfinding strategies. We also explore the impact of different hyperparameters on the performance of the biased decoding strategy. The specific research

[1] Random sampling is the simplest stochastic implementation of a decoding strategy, where each token is chosen according to the exact probability distribution produced by the model at each step.

objectives of this project are as follows:

1. To implement a biased decoding strategy that uses MCMC and TPS to generate text.
2. To evaluate the performance of the biased decoding strategy against a set of benchmarks, including the HumanEval code generation benchmark.
3. To compare the performance of the biased decoding strategy against other decoding strategies, including greedy search, random sampling, and global pathfinding strategies.
4. To explore the impact of different hyperparameters on the performance of the biased decoding strategy.
5. To provide a set of recommendations for future work in the field of text generation and decoding strategies, particularly regarding use of the biased decoding strategy for reinforcement learning.

1.3 Report Structure

This report is structured as follows: **Chapter 2** provides an overview of the related work in the field of text generation and decoding strategies; **Chapter 3** describes the methodology used to implement and evaluate the biased decoding strategy; **Chapter 4** presents the results of the experiments conducted to evaluate the performance of the biased decoding strategy; and **Chapter 5** discusses the implications of the results and provides recommendations for future work.

2 Literature Review

Neural text generation is becoming an increasingly researched topic in the field of natural language processing (NLP). Initial text generation methods employed rule-based systems, utilising pre-defined grammar structures and vocabularies to produce text based on the developers' predetermined rules [9, 10]. While these systems were effective in limited domains, they struggled to produce coherent and diverse text in more general settings.

The advent of LLMs has revolutionised the field of NLP, enabling the generation of human-like text with applications to numerous fields. Language models are now ubiquitous across the wider scope of artificial intelligence (AI) research, with potential applications ranging from drug discovery, image captioning, translation, dataset generation, among others.

2.1 Origins and Evolution of Language Models

The origins of NLP as a field of study can be traced back to the 1950s, when researchers at Georgetown University and IBM demonstrated the first computational approach to language translation [9]. Using a small set of pre-defined grammar rules and vocabulary translations, they showcased a system capable of translating short Russian sentences into English, which was considered a major breakthrough at the time. The 1960s saw the release of *ELIZA* [10], a primitive chatbot that used pattern matching and response templates to mimic basic conversational skills (see Figure 2.1). Although these systems performed well in specific contexts, they faced challenges in generating coherent and varied text for broader applications.

Researchers later began to approach NLP as a statistical problem, with the introduction of the n-gram language model in the late 20th century [12]. This approach introduced the concept of generating text sequentially using trainable probability distributions. However, computational constraints limited n-gram models to processing only a few words at a time, often resulting in incoherent text.

The significant success of AlexNet in 2012 [13] demonstrated the potential of deep neural networks when trained on large-scale datasets, sparking a surge of interest and development in

```
ELIZA: Do you think coming here will help you not to be unhappy?
Human: I need some help, that much seems certain.
ELIZA: What would it mean to you if you got some help?
Human: Perhaps I could learn to get along with my mother.
ELIZA: Tell me more about your family.
```

Figure 2.1: Example conversation with ELIZA, a primitive example of a chatbot developed in the 1960s by Joseph Weizenbaum [10]. Extract source: [11].

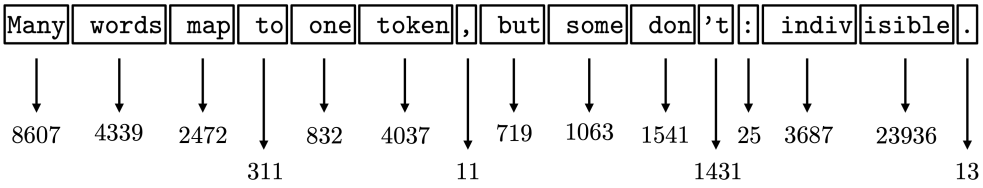


Figure 2.2: Example of text tokenised using the GPT-3.5 tokeniser [16]. Note how whitespaces are included at the start of most tokens for improved efficiency.

the field of deep learning. Initially, researchers struggled to apply deep learning to language models due to the sequential nature of linguistic data, which made training these models computationally expensive and time-consuming. Recurrent neural networks (RNNs) for were experimented with for text generation tasks in the early 2010s[14], but they struggled in terms of inference accuracy and training efficiency [14]. The transformer architecture eventually addressed these issues in 2017 [15], addressed these issues by utilising positional encoding and an attention' mechanism

In 2022, OpenAI released ChatGPT to the public, making the power of these models accessible through a familiar and user-friendly chat interface. This sparked a surge in development and interest, with numerous organizations investing in the advancement of LLMs.

2.2 Fundamentals of Language Models

This section covers the underlying principles of language models, providing a foundation for the subsequent chapters.

2.2.1 Tokenisation and Vocabulary

Language models, like most computational models, inherently work with numerical values. A *tokeniser* function maps text inputs into a series of numerical tokens for the model; following generation, the inverse of the tokeniser maps the newly generated tokens back into human-readable text. In addition to defining the model's vocabulary, the tokeniser acts as a form of compression by combining frequently occurring character sequences.

The tokenisation function must be determined prior to model training, and requires careful consideration of the model's intended capabilities and vocabulary requirements, including support for code generation or international language support[1]. A common method to find a suitable tokenisation mapping is byte-pair encoding (BPE), as shown in Algorithm A.1 (Appendix A.1). The result of this is typically a vocabulary of around 10^4 to 10^5 tokens, with an average token length of 4-5 characters [16], Figure 2.2. It's also common to add additional

[1]Supporting multiple languages has proven to be a challenge for many language models, as the majority of training data is in English. Performance of LLMs often varies depending on the language used to converse with it with a strong bias towards English-based languages [17].

'special tokens', such as end-of-sequence (EOS) tokens for marking the end of a sequence.

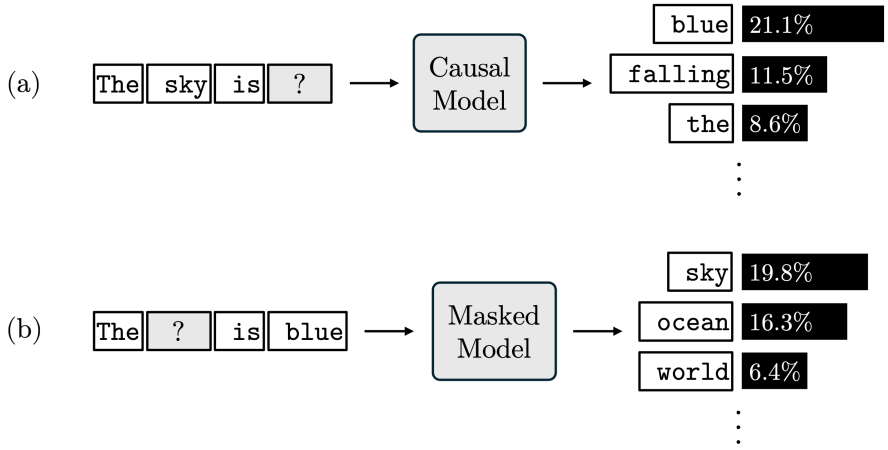


Figure 2.3: The difference between causal (a) and masked (b) language models. With causal models, the model is trained to predict the next token in the sequence, whereas with masked models, the model is trained to predict missing tokens from anywhere in the sequence.

2.2.2 Causal vs. Masked Language Models

Language models can be categorised into two types based on their training objectives. *Causal* language models (CLMs) generate text in order, token-by-token, Figure 2.3a. *Masked* language models (MLMs), however, can generate text anywhere within a sequence, Figure 2.3b. These are sometimes referred to as *autoregressive* and *autoencoding* models, respectively. Although masked models may initially appear more versatile, they have seen less widespread adoption in practice, possibly due to reduced training efficiency [18]. This paper focuses on decoding strategies applicable to causal language models, and henceforth we will use the terms 'LLM' and 'language model' to refer specifically to causal (autoregressive) language models.

2.2.3 Text Generation with Causal Language Models

Neural text generation is typically an iterative process, where the model generates a sequence of tokens one at a time.

A causal language model can be represented as a function which maps an input sequence, $x_{0:t} = [x_0, x_1, \ldots, x_t]$ to a probability mass function (PMF) over the model's vocabulary,

$$\text{CLM} : X \to P(x_{t+1} = v \mid X) \forall v \in V. \tag{2.1}$$

A function of this kind can generate text by predicting a new token, concatenating it to the input sequence, and repeating the process until a stopping criterion is met, outlined in Algorithm 4.

Algorithm 1 Text Generation Process

Input: Tokenised input text, $X = [x_0, x_1, \dots x_t]$
Input: Maximum iterations, N
  $C \leftarrow [\,]$ ▷ Initialise completion as empty sequence
  for $i = 0$ to N **do**
    Run language model with input $[X; C]$ to get probability distribution $P(V)$
    Choose a token $v \in V$ from $P(V)$ according to decoding strategy
    if $v = V_{\text{EOS}}$ **then** ▷ Early stopping criterion
      break
    end if
    $C \leftarrow [C; v]$ ▷ Append token to completion
  end for
  $C \leftarrow$ De-tokenise C ▷ Convert completion from tokens to text
Output: Generated text, C

2.2.4 Prompts and Prompt Engineering

The text generated via this process is, of course, largely dependent on the initial text sequence, commonly referred to as the prompt. Language models are highly versatile, exhibiting the ability to respond to a wide variety of prompts, ranging from simple conversational queries (Figure 2.4a) to complex coding tasks (Figure 2.4b). This versatility allows LLMs to be applied to a wide range of tasks without requiring additional training or fine-tuning to the task at hand.

```
The following is a conversation
between a user and a helpful AI
assistant.
AI: How can I help?
User: [USER QUERY]
AI: [GENERATED RESPONSE]
```

(a) Chatbot prompt

```
The following is a Python function
that takes a list of integers and
returns the sum of the list.

def sum_list(lst):
[GENERATED CODE]
```

(b) Code generation prompt

Figure 2.4: Typical input prompts for different applications of the same language model.

Similarly, the stopping criterion for these models can be adjusted to end once a specific word or phrase is generated. For example, the first prompt in Figure 2.4a could be set to stop once the model generates another instance of "`User:`" to prevent the model from generating an entire conversation by itself.

2.3 Transformer Architecture

The transformer architecture, introduced in 2017 [15], is widely considered one of the most significant developments in the field of NLP since its inception. The performance gains are

largely seen from its scalability, allowing for the training of models with billions or even trillions of parameters [19]. This is achieved via a combination of positional embedding and a self-attention mechanism, enabling many sequences (and their subsequences) to be processed in parallel. All current SOTA language models, including GPT-4 and Llama 3, are based on the transformer architecture [20, 21].

The transformer can be broken into several key components, each of which plays a crucial role in the model's ability to process sequences in parallel: the *embedding layer* (Section 2.3.1) converts the input sequence X into dense vectors, the *transformer layers* (Section 2.3.2) update these vectors according to their context, and the *un-embedding layer* (Section 2.3.3) converts these vectors into the final token probabilities, $P(V \mid X)$.

2.3.1 Word Embeddings and Positional Encoding

Similar to how the tokeniser maps words to numerical values, the embedding layer transforms each token to a high-dimensional vector (the 'word embedding'), Figure 2.5. One can imagine this vector space as a vast area where each token finds its own unique position representative of its semantics; as such, tokens with similar meanings are found close together in this space, and tokens with different meanings are far apart. This is typically achieved by multiplying the input vector(s) with a single learned matrix transformation.

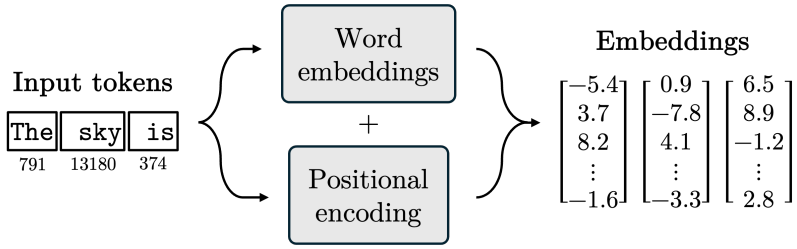


Figure 2.5: Example transformation of an input sequence to high-dimensional word embeddings.

Crucially, this layer also incorporates positional information of the tokens into these embeddings; this is necessary as transformer models are inherently agnostic to the order of input tokens. A pre-defined function maps each position in the sequence to a unique position embedding vector; in the original paper, the authors used a sinusoidal function,

$$\begin{aligned} PE_{(d,2i)} &= \sin\left(d/10000^{2i/d}\right) \\ PE_{(d,2i+2)} &= \cos\left(d/10000^{2i/d}\right) \end{aligned} \tag{2.2}$$

where x = token position, d = vector dimension.

This provides unique positional embeddings for all potential positions in the sequence, which are added element-wise to the corresponding word embeddings. This is crucial for the transformer's ability to process sequences in parallel – sequences from vastly different contexts and lengths can be processed with no change to the underlying calculations or subsequent output, as their relative positions remain consistent under this scheme. One downside of this approach is that

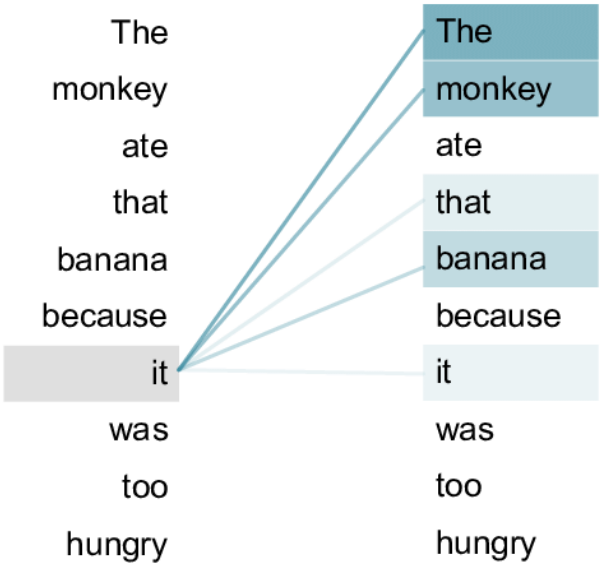


Figure 2.6: Example of the type of relationship an attention layer will use to update the word embeddings in the sequence.

the model must learn the meanings of these positional embeddings, which can be challenging for very long sequences [22].

2.3.2 Attention Mechanism

At the core of the transformer architecture is the self-attention mechanism, which allows the model to learn relationships between tokens in the sequence. As there are many different ways tokens can relate to each other (a basic example being an adjective relating to its noun), multiple attention mechanisms are deployed in parallel, each learning a different type of relationship, such as in Figure 2.6.

This mechanism is implemented as an attention layer, which takes the word embeddings as input and outputs a set of 'attention weights' for each token. These attention weights are used to update the word embeddings according to the prior context of the sequence, Figure 2.7.

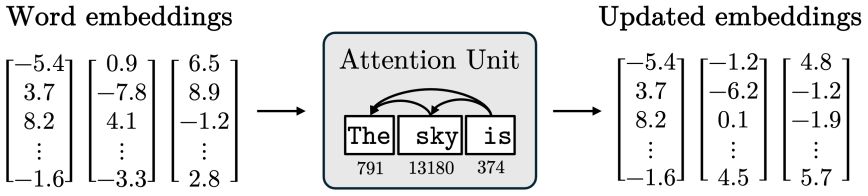


Figure 2.7: Simplified overview of the attention layer in a transformer block.

These layers are usually stacked on top of each other, each learning to extract different features from the input sequence. Multi-layer perceptrons (MLPs) are also applied following each attention layer to further refine the embeddings – this is necessary for the network to learn more complex relationships between tokens, as well as storing factual information derived through the training process [23].

The transformer layers work to progressively condense the entire sequence's meaning into contextualized embedding vectors. Following our example text, each word embedding will contain the entire semantics of "`The`", "`The sky`" and "`The sky is`", respectively.

2.3.3 Un-embedding

The un-embedding layer acts the inverse of the embedding layer, converting the final embeddings of the sequence into probability distributions like in Figure 2.8. This layer can be trained as the inverse of the embedding matrix [24], or in more complex models, may incorporate additional feed-forward layers for more fine-grained predictions.

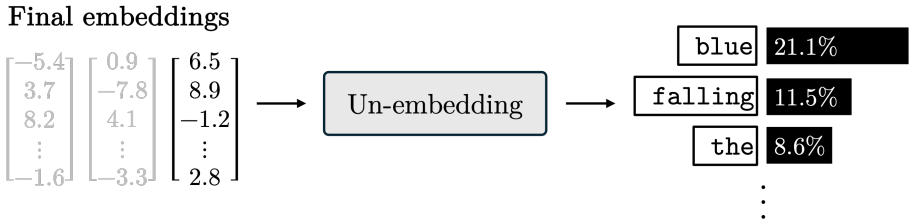


Figure 2.8: The un-embedding layer converts the final embeddings into a probability distribution across the model's vocabulary.

In more advanced models, an additional MLP is applied to the final embeddings to further refine the output probabilities.

2.3.4 Sequence Batching and Parallel Inference

Like many machine learning models, transformers can process large batches of potentially unrelated inputs concurrently to accelerate training and/or inference. In the context of LLMs, this is achieved by inputting a matrix of tokens rather than individual sequences, Figure 2.9. To ensure uniform sequence length within a batch, special 'padding' tokens are inserted where necessary, which the model learns to ignore during processing.

However, transformers also exhibit another type of parallelism: for each forward pass, the model generates predictions for every token in the input sequence simultaneously. This is possible because each word embedding is ultimately transformed into an updated embedding that encapsulates the semantics of the entire preceding sequence. That is to say, when predicting the continuation of `The` `sky` `is`, an LLM will *also* compute predictions for `The` and `The` `sky`. This allows a single input sequence to serve as multiple training examples, significantly improving computational efficiency. During text generation, only the final word embedding is used as in Figure 2.8, as the intermediate predictions are not needed; however, they are still computed as this is an inherent component of the transformer's design.

This is key to the transformer's success: a single batch can effectively provide hundreds of thousands of training examples[2]. This is computed in a single training step, enabling the

[2]For instance, GPT-2 [1] was trained using batches of 512 sequences, each with a length of 1024 tokens.

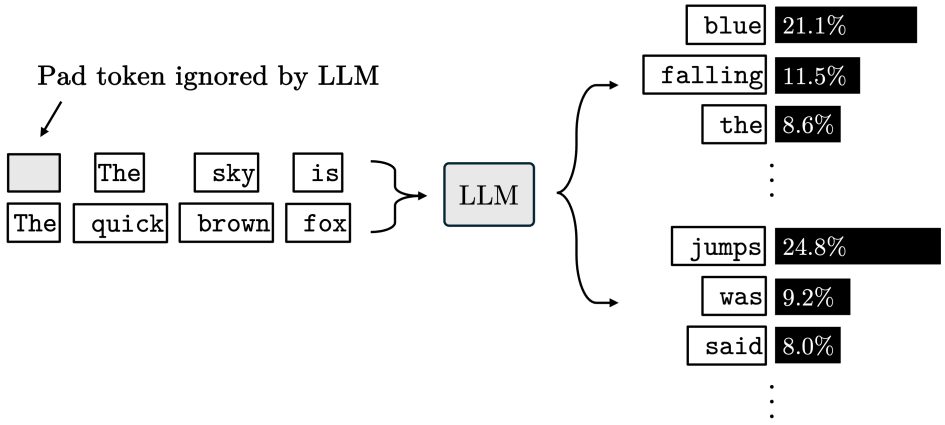


Figure 2.9: Illustration of sequence batching for parallel processing. Note the use of a padding token to equalise the lengths of the sequences. In practice, much larger batches (in the hundreds or thousands) are used to leverage the full computational power of modern hardware [1, 7].

transformer's weights to be trained with a vast amount of data in a short amount of time relative to other architectures. This is, however, naturally dependent on the specific hardware configuration used to train the model.

2.3.5 Training Paradigms

The training of LLMs involves several key paradigms that contribute to their impressive capabilities. These paradigms have evolved alongside advancements in computational resources and our understanding of neural network architectures.

Self-supervised Pre-training

The foundation of modern language models lies in the initial training on vast corpora of text data. This approach was pioneered by models like GPT-1 [25] and BERT [26], which showed that training large-scale transformers on scraped web data allows models to effectively learn language patterns and representations.

The training is performed as follows: the model repeatedly attempts to predict the next token from a sequence of human-written[3] text and its predictions are compared to the ground truth (the actual text). This is considered a 'self-supervised' approach, as it is performed under the typical supervised training paradigm, but does not require any data to be manually labelled like supervised training typically necessitates.

The original GPT-1 was trained using BookCorpus [28], a set of 7,000 unpublished books of various genres; the most recent SOTA models are trained on much larger scales of data. For example, Llama 3 was trained on text corpora totalling over 15 trillion tokens [21]. The sources

[3]Training data can also be synthesised by another LLM. This was the premise for the Phi model series [27], which used synthetic data from larger models to train a smaller model with similar capabilities.

of these datasets are rarely disclosed by the companies that develop the models, but are likely to be the result of mass web-scraping.

The growing development of LLMs has called for an unprecedented amount of resources to be allocated to training bigger models, with the cost of training GPT-4 stated to be over 100 million USD [29]. This has led to a growing concern around the environmental impact of training these models, as the computational resources required to train them are substantial [30].

Supervised Fine-tuning

Following pre-training, models can be fine-tuned on specific tasks or domains to enhance their performance in targeted areas [31]. This process involves further training on a smaller dataset (such as conversations rated positively by users). Fine-tuning can be applied to a wide range of tasks, from sentiment analysis to question-answering, allowing a single pre-trained model to be adapted for multiple downstream applications.

2.3.6 Capabilities and Emergent Properties

LLMs have repeatedly shown proficiency across a wide array of traditional NLP tasks: translation, summarisation, and semantic analysis are all within the scope of these models [1]. The contextual understanding of LLMs is ever-increasing, with recent releases such as Gemini 1.5 [2] able to process entire documents or codebases up to a million tokens in length.

However, the most remarkable quality of LLMs is their ability to generalise to tasks for which they were not explicitly trained. The acquisition of these so-called 'emergent properties' is still not fully understood, and the implications are not fully agreed upon; some see it as evidence of generalised intelligence as a result of the models' scale [3], while others argue that these properties are simply illusions provided by the magnitude of the datasets used [32, 33].

2.3.7 Limitations

While LLMs have proven highly effective at a wide range of tasks, they are certainly not without limitations. For one, they are not immune to the bias seen in many machine learning models, which is a significant issue given that the training data for these models is often sourced from the internet. They may also repeat training material, sometimes verbatim, which leads to concerns about copyright and data privacy.

On the other hand, LLMs are also prone to 'hallucinations' [34] – generating text which *seems* truthful but is in fact entirely fabricated. This is particularly troublesome when asked about recent events, as the highest quality response from its training may be incorrect or outdated.

Tokenisation also introduces certain challenges. For instance, it is difficult to fairly train a model's multilingual capabilities, as the tokenisation mapping is likely to be biased towards efficiency in English [17]. It also introduces a level of abstraction between the model and the text;

current SOTA models may also struggle with seemingly basic tasks like counting the number of words in a sentence or the number of characters in a word, as this information is stripped by the tokeniser.

Finally, the sheer size of these models can be a significant barrier to their adoption. Training a model like GPT-4 can cost millions of dollars in computational resources, and the hardware required to run these models in production can be prohibitively expensive [29].

2.4 Language Model Decoding Strategies

Decoding strategies are integral to the neural text generation process. The choice of decoding strategy can significantly impact the quality of the generated text, as well as the computational resources required to generate it. In this section, we explore a variety of decoding strategies for neural language models, and differentiate between the different types of strategy.

As discussed in Section 2.2.3, the language model generates probabilistic data for a given input; the decoding strategy is responsible for converting this data into a coherent output. This process can be deterministic, leading to the same output every time, or stochastic, leading to potentially different outputs for the same input.

2.4.1 Statistics of Language Generation

It is beneficial to first define some key concepts in probability theory, as they are essential to understanding the role of decoding strategies in text generation. A probability mass function (PMF) is a function which maps each element of a set Z to a value $f(z)$, such that

$$f(z) \geq 0 \ \forall z \in Z \quad \text{and} \quad \sum_{z \in Z} f(z) = 1 \tag{2.3}$$

where f is a PMF over the sample space Z. In the context of language models, a decoding strategy is like a PMF over the space of sequences which could follow on from the input sequence. If we denote the model's unaltered output at each step as a PMF, $P(v \mid X)$, then the decoding strategy acts as a filter of P to produce a transformed PMF, denoted $\tilde{P}$.

An important caveat is that LLMs themselves do not necessarily output a PMF, but rather unscaled scores or *logits* for each token in the model's vocabulary. These are typically passed through the softmax function,

$$\sigma(Z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}, \tag{2.4}$$

where $Z = [z_0, z_1, \ldots, z_n]$. This scales the logits to create a suitable PMF that satisfies the conditions defined in 2.3. The PMF which arrises from random sampling is thus $P(V) = \sigma(Y)$. Decoding strategies can either act on the logits directly, or on P; in either case, the strategy acts as a transformation of $P \to \tilde{P}$.

2.4.2 Local Decoding Strategies

The simplest implementation of a decoding strategy is *greedy search*, in which the highest-scoring token is chosen at each step. This is naturally the most efficient strategy in terms of computation, but can lead to suboptimal results for two main reasons.

For one, human language is naturally very stochastic in nature [6]; for text to appear natural, it needs to reflect this. Greedy search, however, tends to produce repetitive and uninteresting responses as it is confined to choosing the most likely (thus most predictable) token at each step.

Further, always choosing the most likely token leads to insufficient exploration of the search space, leading to the text getting 'stuck' in a local minima (sometimes referred to as 'text degeneration') [6]. An example of this can be seen in Figure 1.2a. This arises as the likelihood of pre-occured tokens is reinforced, leading to a positive feedback loop once a word or phrase has already been repeated.

To remedy these issues, it is common to use a stochastic decoding strategy, which introduces an element of randomness to the model's output. However, this introduces a new problem of balancing accuracy and diversity in the model's output. There are a few ways of balancing this trade-off:

1. **Temperature Sampling**: The entropy of the probability distribution can be controlled by dividing the model's output logits by a temperature parameter, $\tau \in \mathbb{R}^+$, before applying the softmax function (2.4) for the resulting PMF. This allows for control over the diversity of the model's output, with higher values of τ leading to more diverse responses, and lower values leading to more peaked probability distributions, depicted in Figure 2.10.

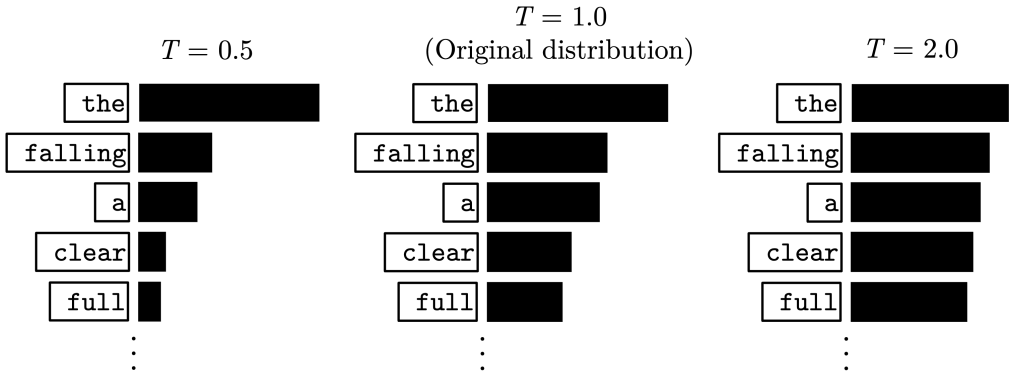


Figure 2.10: Example effects of temperature on the output distribution of a language model with $T = 0.5$, $T = 1.0$ and $T = 2.0$.

   Values between 0 and 1 are typically chosen, effectively reducing the probability of selecting lower-probability tokens while still maintaining a degree of diversity.

2. **Top-k Sampling**: Considers only the $k \in \mathbb{Z}^+$ most likely tokens at each step, and samples from this subset according to their probabilities, shown in Figure 2.11. This is achieved by

setting the logit value of tokens outside of the top-k to $-\infty$, redistribution the probability mass between the chosen tokens. Naturally, if top-k sampling is performed with $k = 1$, the output will match a greedy search decoder.

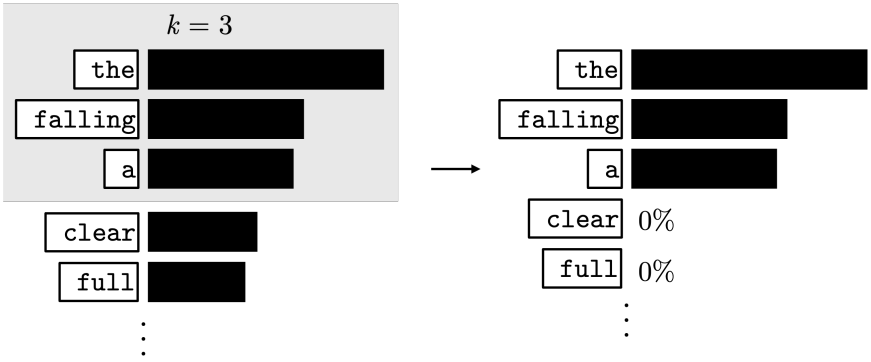


Figure 2.11: Example effect of top-k sampling on the output distribution of a language model; in this example we use $k = 3$.

3. **Top-p (Nucleus) Sampling**: Similar to top-k, but dynamically adjusts the number of tokens considered at each step to match a cumulative probability value p, where $p \in \mathbb{R}^+ \leq 1$. In the original paper that introduced this strategy [6], the authors found that $p = 0.95$ resulted in text with similar probabilistic properties to human text and thus is a common choice for this parameter.

Text generation using local strategies can be implemented using Algorithm 4 from Section 2.2.3. The main advantage of local sampling strategies is that they can provide a diverse range of outputs, and each output can be yielded token-by-token as the model generates it. This has made them a popular choice for chatbots and AI-based assistants, where minimising the delay to the first token is preferable. However, they are blind to the future context of the text, thus leading to sub-optimal results.

2.4.3 Global Decoding Strategies

To fully leverage the power of a language model, it is sometimes beneficial to perform a global search on the output space, selecting an entire output sequence rather than individual tokens sequentially. The obvious drawback to this is added computational complexity, as the search space grows exponentially with the length of the output sequence.

A simple example of a global strategy is *best-of-n* decoding, where n full samples are generated and the best is chosen according to the final joint probability (or some other metric). This can be useful for tasks which require a specific property in the output, such as for human preference alignment [35], where fine-tuning the underlying model is not an option.

A more common global strategy is *beam search*, which borrows from graph theory. The algorithm maintains a list of the $B \in \mathbb{Z}^+$ most likely sequences at each step, and continues to expand these sequences until the end of the sequence is reached, Figure 2.12. Following a beam search, the sequence with the highest overall probability is chosen as the model's output. The

beam width, B, is a hyperparameter which controls the number of sequences which are expanded at each step. A beam width of 1 is equivalent to a greedy search decoder, while larger values of B can lead to more diverse and accurate outputs. B^2 sequences are considered in total, making it a very expensive strategy. It is a deterministic strategy, making it unsuitable for tasks which require a high level of diversity.

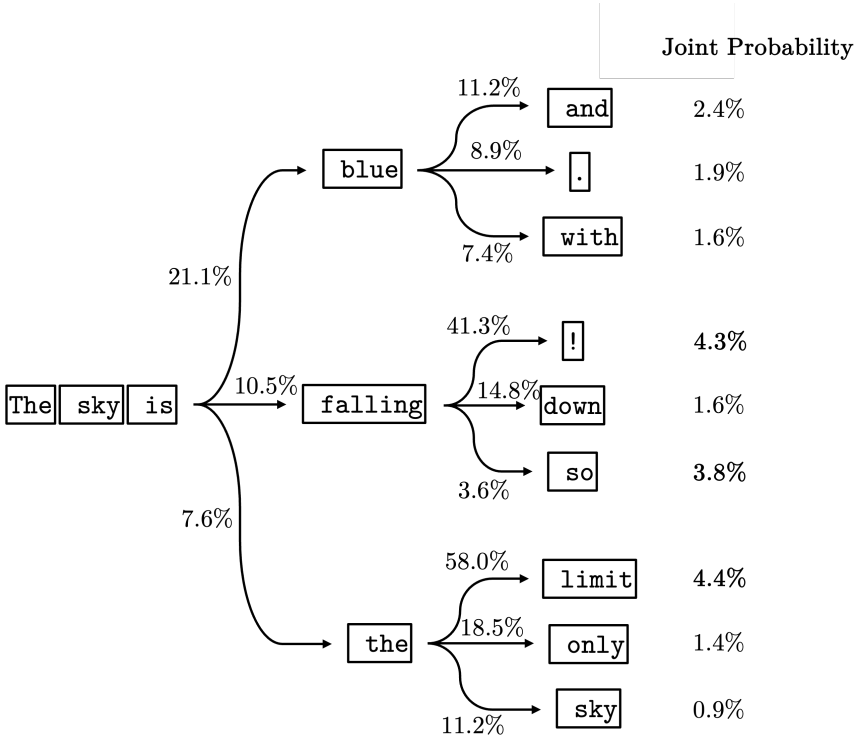


Figure 2.12: Example of beam search for neural text generation with a beam width of 3.

2.4.4 Relationship to Output Quality

Different decoding strategies can utilise a language model's capabilities in different ways. For coding tasks, where accuracy is favoured to diversity, a deterministic strategy such as greedy search is often preferred. For chatbots or creative writing, it is common to use top-p sampling, as this has shown to produce the most human-like text given a suitable value of p is chosen [6].

However, local sampling strategies are limited by their lack of broader context, inherently increasing the likelihood of producing incorrect or incoherent text due to the sampling process. This becomes more apparent in tasks requiring high levels of factual accuracy or logical reasoning, where the noise introduced by the sampling can often derail the model's output [36].

Global strategies work to resolve this issue by considering the sequence as a whole; however, these are also not without their drawbacks. As it only considers the top B tokens at each step, beam search can still get stuck in local minima [37]. Further, [37] noted that beam search tends to unnaturally favour shorter completions, as the probability of the sequence

Table 2.1: Summary of decoding strategies

Decoding Strategy	**Parameter(s)**	**Description**
Greedy search	None	Highest-scoring token at each step
Random sampling	None	Randomly samples from $\sigma(Y)$ at each step
Temperature sampling	$\tau \in \mathbb{R}, \tau > 0$	Emphasises more likely tokens as $\tau \to 0$. Temperature sampling with $\tau = 1$ is equivalent to random sampling.
Top-k sampling	$k \in \mathbb{Z}, k > 0$	Samples from top-k tokens at each step
Top-p sampling	$p \in \mathbb{R}, p > 0$	Samples from smallest combination of tokens that satisfies $\sum P(v) \leq p$ at each step
Beam search	$B \in \mathbb{Z}, B > 0$	Expands B most likely sequences at each step, picks best sequence overall
Best-of-n	$n \in \mathbb{Z}, n > 0$	Picks highest of n sequences generated via random sampling

decreases exponentially with its length. This can lead to the model generating text which is too short for the given context.

Best-of-n is far less common. Unlike the other decoding strategies which rely on the probability distribution of the model's output, best-of-n can work to bias outputs towards any objective function. As such, it has found nuanced use-cases for tasks which require certain properties in the output text without fine-tuning the model itself, such as for human preference alignment [35]. It's unclear how well best-of-n would alter the quality of the model's output, as the value of n may need to be very large to ensure effects are seen.

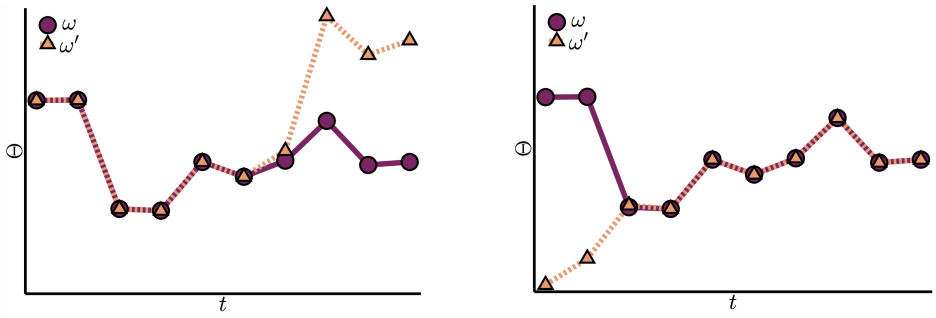
2.5 Transition Path Sampling

2.5.1 Background: Monte Carlo Algorithms

Monte Carlo (MC) algorithms employ random sampling techniques to obtain numerical results for problems that are infeasible to solve directly. The core concept behind MC methods is to use random sampling to estimate the value of a function or the probability of an event by generating many random samples and averaging the results. These methods are applied to problems that are challenging or impossible to solve analytically, such as evaluating complex integrals or simulating physical systems with numerous degrees of freedom.

2.5.2 Sampling Rare Events

However, some events occur with too low of a probability to be observed directly through an MC approach. These are called 'rare events' – state transitions that are so infrequent that attempting to observe them directly would be infeasible, even in simulation. One example of this is an unfolded protein spontaneously folding on itself. If one wanted to study the dynamics



(a) Forwards shooting ($\kappa = 1$) (b) Backwards shooting ($\kappa = -1$)

Figure 2.13: Examples of shooting TPS algorithm which can work either forwards or backwards. *Image source: [39]*

of this event in simulation, it could take an extremely long amount of time for such an event to occur through random sampling, so an MC approach would not be viable. Of course, one could alter the dynamics of the simulation to increase the likelihood of such an event occurring, but this would likely ruin the integrity of the results.

Transition path sampling (TPS) can be used to explore these events as if they occurred more likely while respecting the dynamics of the original system [38]. Rather than simulating the entire system until a transition occurs, TPS decomposes the transition into smaller components (e.g., individual bond-breaking and bond-forming events in protein folding), which are then individually sampled to construct a complete transition path. This can be repeated to generate an ensemble of potential transition paths between the two states. As the underlying probabilities of the system are unaltered, the transitions can then be analysed as if they had occurred 'organically' (for example, to study the reaction mechanism).

2.5.3 Shooting Algorithm

TPS is typically implemented using a *shooting algorithm* [8], which samples these rare transition paths more efficiently than typical Monte-Carlo algorithms. It does this by taking a random point in time, t, and 'shooting' a new sample from that point, which is accepted with a probability proportional to its time-delta interval. This procedure can be done either forwards or backwards in time, depicted in Figures 2.13a and 2.13b respectively.

Algorithm 2 Original Shooting TPS Algorithm [8]

Input: Starting trajectory Θ
Output: Updated trajectory Θ'

$\Theta' \leftarrow \Theta$ ▷ Initialise candidate trajectory
$t \leftarrow$ Choose random integer from $[1, \tau]$
$\kappa \leftarrow$ Choose random direction from $-1, 1$
while $t + \kappa \geq 1$ and $t + \kappa \leq \tau$ **do**
    $\Delta\Theta \leftarrow$ Choose random alteration to trajectory
end while

2.5.4 Applications in Machine Learning

The shooting TPS approach has shown to be effective as a strategy for training ensembles of neural networks [39]. This works as the shooting algorithm was able to effectively explore the search space of parameters, leading to an ensemble of networks with diverse (yet equally effective) parameters.

2.6 Evaluation Methods

With the growing popularity and competition in the field of language models, a variety of evaluation methods have been developed to compare the performance of LLMs across various domains. The majority of benchmarks work to evaluate the model's accuracy – tasks like code generation, factual recall, or mathematical problem solving – as these have clear methods of evaluation. Evaluating the model's ability to generate coherent text is naturally more challenging, as it is a subjective task with less rigid criteria.

While many benchmarks are designed for evaluating the model's raw performance, some may be adapted for evaluating decoding strategies using the same underlying model.

2.6.1 Perplexity Score

Even prior the advent of LLMs, perplexity (PPL) has been a common measure of a statistical model's performance, which has recently found use in evaluating language models. Its original definition [40] is the exponentiation of the entropy of the model's output, defined as

$$PPL(p) = \exp\left[-\sum_{x} p(x) \log p(x)\right]. \tag{2.5}$$

Lower perplexity scores indicate a more confident model, as its output entropy is lower (one class is predicted to be far more likely than the others). This has been adapted to language models [41] by calculating the model's perplexity across a given test set of text and normalising by the number of tokens in the text,

$$PPL(X) = \exp\left[-\frac{1}{T}\sum_{t=1}^{T} \log p(x_t \mid x_{0:t-1})\right]. \tag{2.6}$$

While this metric is typically used to benchmark the models themselves, it can be adapted to evaluate decoding strategies. By generating corpuses of text with a given decoding strategy, we can calculate the model's perplexity (or the joint log probability) using the model's original probability distribution P at each step. From this we can analyse the diversity and accuracy of each decoding strategy relative to random sampling.

2.6.2 Code Generation Tests

Code evaluation is a relatively new concept as language models have only recently become advanced enough to produce reliable, working code. Evaluating a model's ability to generate

```
from typing import List def has_close_elements(numbers: List[float],
    threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to
    each other than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True """
    ...
```

Figure 2.14: Example prompt from the HumanEval dataset [42].

working code is a relatively straightforward task, as the code can be run and tested automatically to check its validity. One such benchmark is HumanEval [42], a suite of coding tasks with automated tests to evaluate the functional correctness of neurally generated code. It consists of 164 human-written programming tasks in the Python programming language, along with a series of tests for each task.

Each task is presented as the beginning of a Python function, accompanied by a brief description and expected outputs, as illustrated in Figure 4.8. The language model then generates code until one of a pre-defined list of 'stop phrases' are generated[4], at which point the task ends. Once all tasks have been attempted, the provided tests can be used to generate the relevant evaluation metrics.

The paper also defined the useful *pass@k* metric, as the fraction of solutions where any of k solutions are correct. This is especially beneficial for evaluating smaller or weaker models where a single solution is unlikely to pass – for example, in the original paper, they evaluated GPT-3 as scoring 10.0%, 30.0% and 70.0% for pass@1, pass@10 and pass@100 respectively. Recently, pass@1 scores have become the primary reported metric for new models, as they better represent real-world scenarios where first-attempt success is crucial, and most models now achieve moderate scores with single solutions. Typically, many more than k solutions are produced, so that the value can be more accurate. This benchmark is typically run in a zero-shot fashion, meaning that no completed examples are provided in the prompt.

2.6.3 Massive Multitask Language Understanding

The *Massive Multitask Language Understanding* (MMLU) benchmark was introduced in 2020 [43] to test language model accuracy across a wide variety of domains, with over 14,000 questions split into 57 subjects such as mathematics, computer science, world history, and many others. Each task is a simple multi-choice question whereby 3 incorrect choices and 1 correct choice are provided. Each choice is marked A, B, C or D and the model can then choose one of these

[4]The paper defines their stop words as "`\nclass`" and "`\ndef`", among other phrases which are commonly generated by the language model following the completion of the function. It is important that generation is stopped at this point to save compute and prevent irrelevant code generation outside of the scope of the task.

options with a single token. The prompt typically contains 5 completed examples related to the task before the uncompleted task is presented in the prompt, in order to guide the model into the correct answering format. This, in conjunction with the single-token response basis, ensures that the model response style is inconsequential to the results.

2.6.4 MAUVE Score

Introduced in 2020, the MAUVE score [44] is a metric designed to evaluate the quality and readability of text generated by language models. It is calculated by measuring the similarity between generated text and a (human-written) reference text, with higher scores indicating a higher degree of similarity.

3 Methodology

This chapter outlines the theoretical framework behind our methodology, the implementation of the TPS-based decoding strategy, and the specific metrics used to evaluate the performance of the decoding strategies.

3.1 Theoretical Framework

3.1.1 Defining Completion Quality

Recall from Section 2.6.1 that perplexity, in the context of LLMs, is a measure of the model's 'certainty' when processing a particular test text (2.6) [41]. While this is typically used for evaluating a model using a known text corpus, we can certainly utilise a similar metric to evaluate the model's confidence in one of its own completions. For a given completion, we can calculate the log of its joint probability normalised by the length of the completion,

$$Q(C \mid X) = \frac{1}{m} \log P(C \mid X) = \frac{1}{m} \sum_{i=0}^{m-1} \log P(c_i + 1 \mid [X; c_{0:i}]), \tag{3.1}$$

where X is the prompt, and C is the completion with a length of m tokens. We shall refer to this metric as the 'quality' of the completion in order to avoid confusion with the probability values P and $\tilde{P}$. Higher quality values should indicate that the model believes the sentence to be more likely to appear, with lower values indicating that the sentence is unlikely to appear. Table 3.1 provides some example sequences with their relative quality scores, assessed using GPT-2 [1].

This could reveal the diversity of the text produced with a given strategy, as well as the accuracy of the strategy. This is a useful metric for understanding the transformation of the model's output distribution by different decoding strategies, as well as the quality of the text produced by each strategy.

Table 3.1: The 'quality' of various completion, defined as the log of the joint probability of the appended tokens, normalised by the completion length.

Input X	**Completion** C	**Quality** $Q(C \mid X)$
The sky is	blue with fluffy white clouds	-3.37
The sky is	turning pink as the sun sets	-2.35
The capital of France is	Paris	-3.43
The capital of France is	home to the Eiffel tower	-1.87
The capital of France is	blue with fluffy white clouds	-5.68
The sky is	Paris	-5.39

3.1.2 Decoding Strategies as Pathfinding Algorithms

For a given input X, we can imagine some 'optimal' completion with the highest possible quality, $C^* = \text{argmax}_C\, Q(C \mid X)$. Under this, decoding strategies can be conceptualised as path-finding algorithms which seek to find C^*. For a vocabulary size n and completion length m, the total search space is n^m completions.

A local strategy alters the PMF at each step, $P \to \tilde{P}$. For example, the PMFs for the greedy search and temperature sampling strategies at each step are given by

$$\tilde{P}_{\text{greedy}}(v) = \begin{cases} 1 & \text{if } P(v) = \max P(v) \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

$$\tilde{P}_{\text{temperature}}(v) = \sigma\left(\frac{Y}{\tau}\right)_v \tag{3.3}$$

respectively. This has a combined effect on the quality of the generated sequences, depicted in Figure 3.1.

Global strategies are inherently better at this search, as they can explore multiple paths simultaneously. Beam search does this at the cost of diversity; the best-of-n strategy suffers from increased computational complexity as each trajectory is explored independently[1].

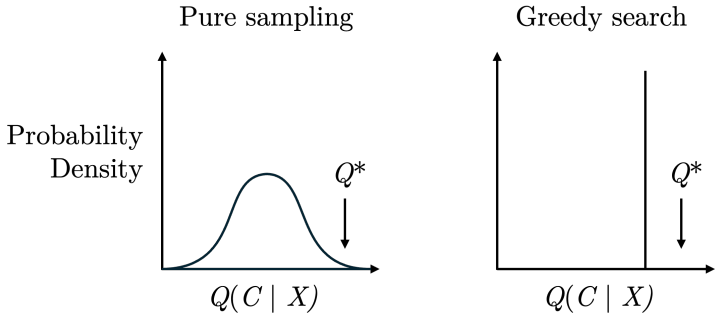


Figure 3.1: Caption

3.2 Application of TPS as a Decoding Strategy

We used the shooting TPS algorithm from Section 2.5.3 to implement a decoding strategy which explores the search space of potential text completions with a bias towards higher-quality sequences. The original algorithm allows for shooting in both directions, however, language models naturally only generate text forwards; as such, we implemented a forward-shooting approach, Algorithm 3.

We hypothesised that this strategy would be effective for text generation, as it allows for exploring the search space of potential text completions with a bias towards higher-quality sequences, similar to how it has been used to sample the parameters of neural network ensembles

[1]We found the best-of-n strategy particularly challenging to implement. Tests frequently had to be run with a smaller batch size to avoid memory issues, a problem not encountered with any other strategy.

Algorithm 3 Forward-shooting TPS for Neural Text Generation

$C \leftarrow$ Initial completion generated with random sampling
for range $1 \rightarrow N$ **do**:
  $t' \leftarrow$ Random position within the generated tokens $[0, m]$
  $C' \leftarrow$ Copy of C
  $C'_{t':m} \leftarrow$ Regenerate tokens from point t', prompt $[X; C_{0:t'-1}]$
  $\alpha \leftarrow \exp\{\beta\,[Q(C' \mid X) - Q(C \mid X)]\}$ ▷ Calculate acceptance rate as ratio of qualities
  $u \sim U[0, 1]$
  if $u < \alpha$ **then** ▷ Accept candidate with probability α
    $C \leftarrow C'$
  end if
end for

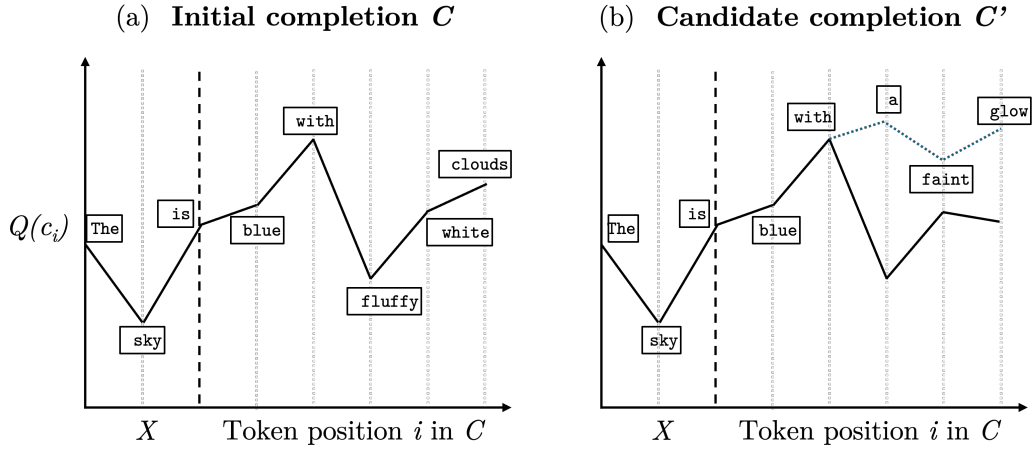


Figure 3.2: TPS Example

[39]. The β parameter provides a way to control the exploration-exploitation trade-off of the algorithm with higher values analogous to exploitation in RL, while a lower β value allows a more varied exploration of the search space. This allows for a more fine-grained control over the algorithm's behaviour, and we can use this parameter to tune the algorithm to find the optimal sequence X^*. More exploration may be beneficial for generating creative and diverse text, while exploitation work better for generating coherent text. Regardless, the algorithm explores the search space in a way that is less greedy than beam search (disregarding any completions outside of the beam width) and more strategic than best-of-n (which acts similarly to TPS, but with $\beta = \infty$ and $t' = t$).

However, performance may be limited by computational constraints. The algorithm is typically run for a huge number of steps; however, computational restraints limit the number of steps we can run to 1,000 for GPT-2 based benchmarks and 100 for Phi-1 coding-based benchmarks. This is a limitation of the computational resources available to us, and we acknowledge that the results may be different with more computational power and/or time.

3.3 Comparative Framework

To cover a wide range of decoding strategies, we implemented the following strategies:

Decoding Strategy	**Parameter Range**	**Range Justification**
Greedy search	-	Baseline strategy
Random sampling	-	Baseline strategy
Temperature sampling	$\tau = 0.0^{\dagger}, 0.2, 0.4, 0.6, 0.8, 1.0^{\ddagger}$	Values above 1.0 are rarely utilised as it exemplifies the negative effects of sampling
Top-k sampling	$1^{\dagger}, 2, 3, 4, 5, 10$	Covers wide range of values; $k >$ 10 is unlikely to have significant effect.
Top-p sampling	$p = 0.0^{\dagger}, 0.5, 0.9, 0.99, 0.999, 1.0^{\ddagger}$	The original paper [6] suggests $p = 0.95$ is a good starting point, so we chose values around this.
Beam search	$B = 1^{\dagger}, 2, 3, 4, 5, 10$	Covers wide range of values; $b >$ 10 is computationally expensive and likely to run into early stopping issues [37].
Best-of-n	$n = 1^{\ddagger}, 2, 3, 4, 5, 10$	Covers wide range of values; $n >$ 10 is computationally expensive.
Biased TPS	$N = 0^{\ddagger}, 10, 100$ $\beta = 0^{\ddagger}, 0.1, 1, 10$	Values of β chosen to cover a wide range of exploration-exploitation trade-offs; compute limitations prevent further exploration of N

†Equivalent to greedy search

‡equivalent to random sampling

Table 3.2: Chosen decoding strategies and their parameter ranges

Text with local decoding strategies was generated according to Algorithm A.2, Appendix A.1; our approach calculates $P(V)$ and samples a new token $v \sim \tilde{P}(V)$ using the same model output, greatly increasing the efficiency of data collection. Global strategies were more complex to implement due to the added dimensions of tokens and probabilities. We implemented beam search and best-of-n as classes which maintain a list of active sequences and their associated probabilities, with methods to update the list and return the best sequence when the generation process is complete. Batches were combined and inferred in parallel where possible to increase generation speed.

Algorithm 4 Text Generation Process

Input: Tokenised input text, $X = [x_0, x_1, \dots x_t]$
Input: Maximum iterations, N
  $C \leftarrow [\,]$ ▷ Initialise completion as empty sequence
  $p \leftarrow 0$ ▷ Completion starts with probability 100%, thus 0
  for $i = 0$ to N **do**
    Run language model with input $[X; C]$ to get probability distribution $P(V)$
    Choose a token $v \in V$ from $P(V)$ according to decoding strategy
    if $v = V_{\text{EOS}}$ **then** ▷ Early stopping criterion
      break
    end if
    $C \leftarrow [C; v]$ ▷ Append token to completion
    $p \leftarrow p + \log P(v)$ ▷ Add log softmax of chosen token
  end for
  $C \leftarrow$ De-tokenise C ▷ Convert completion from tokens to text
Output: Generated text, C
Output: Joint log probability p, equal to $Q(C \mid X)$

3.3.1 Evaluation Metrics

For comparison of the TPS decoding strategy with contemporary strategies, we focused on three core evaluation metrics: joint probability distributions, using the quality metric defined in **??**; code generation as per the HumanEval benchmark; and qualitative analysis of text generated with each strategy.

Probability Distributions

To analyse the distribution of trajectories generated with each decoding strategy.

Joint probability distributions, a.k.a. the quality of the generated sequences as defined above. For this we opted for the 124-million parameter version of GPT-2 [1]. While significantly less capable than the current SOTA language models, we reasoned that using a smaller model would enable us to generate vast amounts of data with a wide scope while being representative of the effects different strategies would have on a larger model. We sampled 1,000 completions for each strategy with a maximum completion length of 50 tokens (excluding the length of the input) and calculated the quality of each completion using Equation 3.1.

All completions used the same input prompt, Figure A.31; hence, our results reflect the distributions produced from the same initial state.

HumanEval

We ran each strategy on the HumanEval code generation benchmark [42], which consists of 164 coding tasks. We generated 10 samples for each task and used the provided tests to calculate a

```
In a shocking finding, scientist discovered a herd of unicorns living
in a remote, previously unexplored valley, in the Andes Mountains.
Even more surprising to the researchers was the fact that the unicorns
spoke perfect English.
```

Figure 3.3: The input prompt used to generate completions for our quality analysis of each decoding strategy. This prompt was initially used to showcase the capabilities of GPT-2 [1], and has since become a popular choice for testing the creative writing of language models.

pass@1 and pass@10 score for each decoding strategy. The pass@1 score is the average proportion of samples that pass the test on the first attempt, while the pass@10 score is the proportion of samples with *any* viable solution after 10 attempts. This pass@k metric allows for a more lenient evaluation of the model's performance, favouring diversity of solutions over accuracy alone. As we could only run 10 samples for each coding task, the pass@10 metric may be less reliable than pass@1, but is included nonetheless.

3.4 Additional Experiments

We also conducted a series of experiments to explore the impact of different hyperparameters on the performance of the biased TPS decoding strategy. We varied the number of TPS steps N and the bias parameter β to see how these parameters affect the quality of the generated sequences. For this, we generated 100 samples for each combination of hyperparameters and calculated the quality (joint probability) of each sequence as described above. These could then be plotted, e.g. in a heatmap, to visualise the impact of each hyperparameter on the quality of the generated sequences. This could also provide insight into the optimal hyperparameters for the biased TPS decoding strategy.

4 Results

4.1 Quality and Probability Distributions

Here we compare the probability distributions produced by each decoding strategy. $Q(C \mid X)$ is our quality metric from Section **??**, which we use to evaluate the quality of the generated completions. Probability density histograms are plotted for each strategy in Figures 4.1–4.6.

A full table, containing the mean (μ) and standard deviation (σ) values for each strategy, is also provided in Table A.1. Refer to Section **??** for a detailed explanation of the quality metric.

Greedy Search and Random Sampling

These results serve as the baseline for comparison with the other strategies. Greedy search produces completions with a mean quality of -1.46 and a standard deviation of 0.00. This is expected, as greedy search is deterministic and thus will always output the same result. Random sampling, on the other hand, produces completions with a mean quality of -3.63 and a range of quality scores between $Q \approx -5$ and $Q \approx -3$.

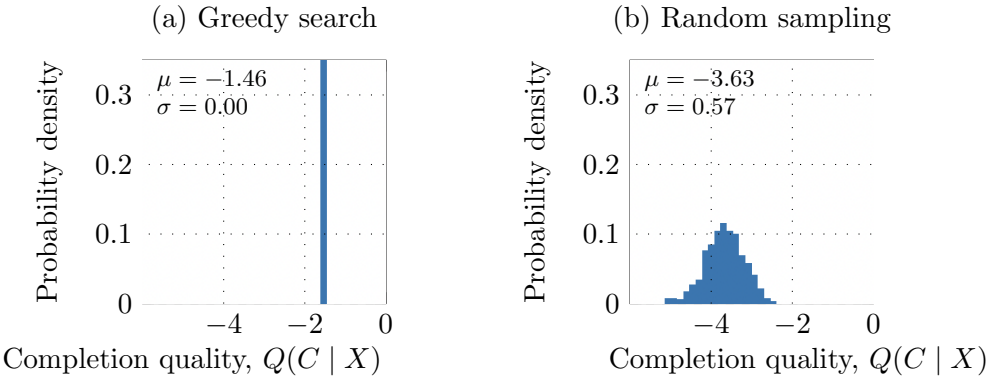


Figure 4.1: Probability distributions of text generated with baseline strategies (greedy search and random sampling)

Temperature Sampling

Temperature sampling allows for control over the entropy of the sampling process at each step. As the temperature parameter τ increases, the distribution of completions becomes more uniform, with $\tau = 1$ corresponding to random sampling. We observe this trend in Figures 4.2a–d, where the mean quality decreases as τ increases.

The effects are more subtle at $\tau = 0.2$ and $\tau = 0.4$, with the same 'spike' in the distribution at $Q \approx -1.5$ found in the greedy search distribution (Figure 4.1a). It isn't until $\tau = 0.6$ that

the distribution shifts significantly from this specific value other completions are more properly explored. As expected, the mean and standard deviation both increase with higher values of τ. The shapes of the distributions appear similar to the original, however increasingly condenced as τ increases.

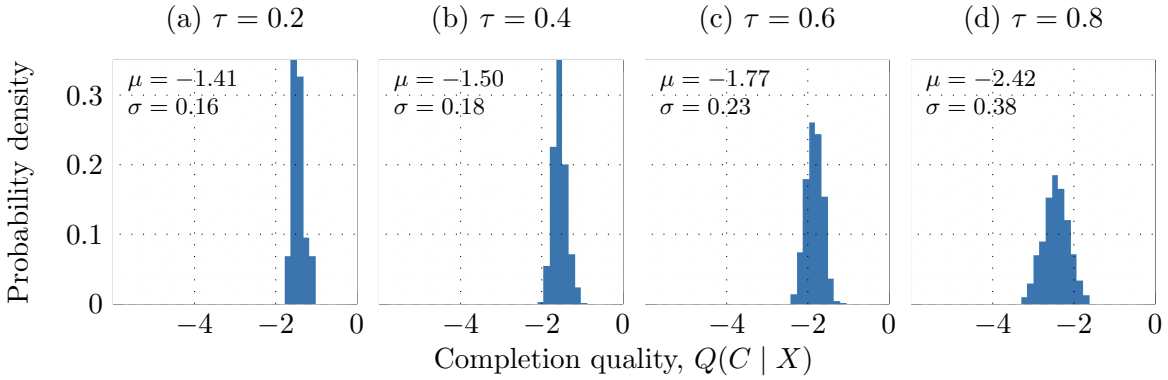


Figure 4.2: Probability distributions of text generated with temperature sampling

Top-k Sampling

Top-k sampling aims to improve the quality of completions by restricting the set of tokens considered at each step. We observe this strategy produces completions with a higher mean quality than random sampling, with a positive skew that increases with k.

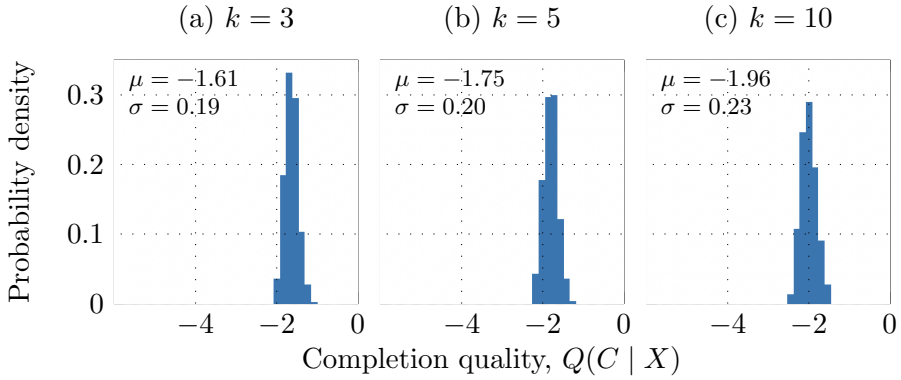


Figure 4.3: Probability distributions of text generated with top-k sampling

Top-p Sampling

Top-p cuts the tail-end of the probability distribution at each step. We see a value of $p = 0.5$ produces similar results to $k = 10$, and $\tau = 0.6$, with a spiked distribution that falls between greedy search and random sampling in terms of both mean and diversity.

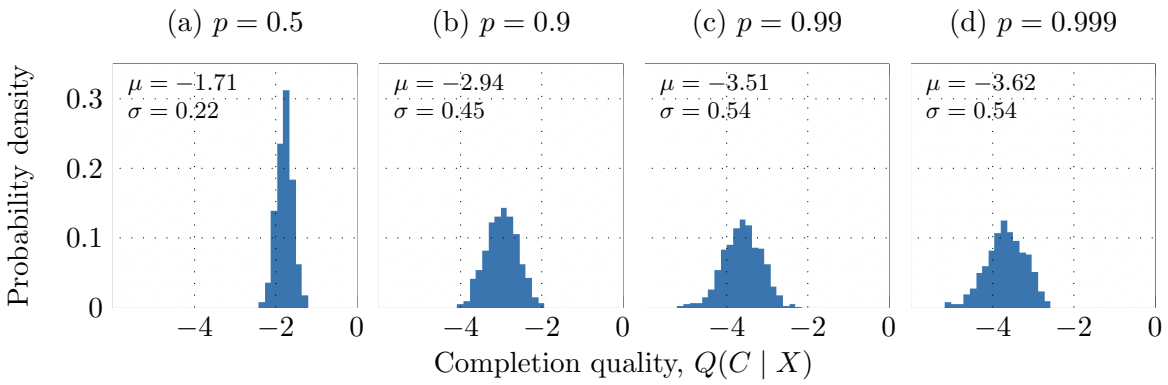


Figure 4.4: Probability distributions of text generated with top-p sampling

Beam Search

As beam search considers multiple paths, it is able to find high-quality sequences that are sometimes missed through greedy token decoding. This is reflected in our results, as increasing the beam width B leads to marginally higher quality sentences. There is of course no variance in the distribution as beam search only ever returns the highest quality sequence from its search.

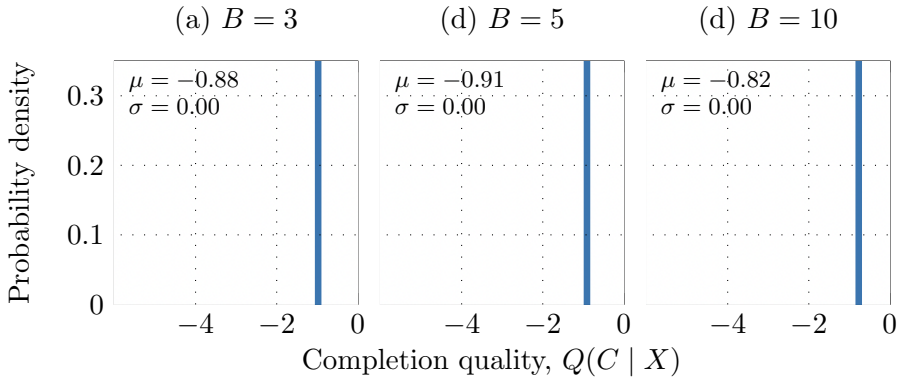


Figure 4.5: Probability distributions of text generated with beam search

Best-of-N

We see that best-of-n effectively improves the mean quality of completions, as well as increasing the diversity with higher values of n. This could be indicative of a strategy which only temperementally finds better results than random sampling.

We also see a completion arise around $Q \approx -5$ – perhaps a sequence which ends early, which is preferred by the strategy as it has a lower total joint probability, but once normalised is lower.

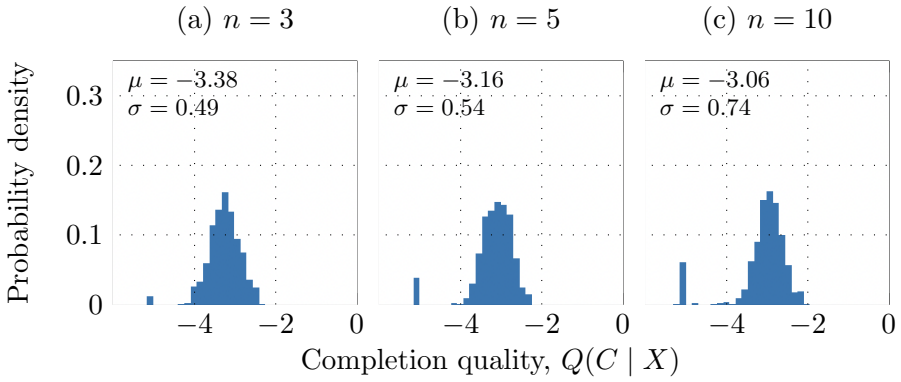


Figure 4.6: Probability distributions of text generated with best-of-n

Shooting TPS

The shooting TPS strategy, implemented at $N = 100$ iterations (Figures 4.7c–d), with a snapshot of results taken at $N = 10$, (Figures 4.7a–b) respectively. We see in both cases that the differences between $\beta = 1$ and $\beta = 10$ are marginal, with almost unchanged mean and standard deviation values. Between $\beta = 0.1$ and $\beta = 1$ there is a shift of the distribution towards higher-quality sentences, with no noticable impact to the diversity of trajectories generated.

A substational change in distribution is seen when increasing the number of iterations of the shooting TPS algorithm. The mean and standard deviation both increase beyond random sampling, suggesting the process is working to explore the search space in the region of higher-quality sequences.

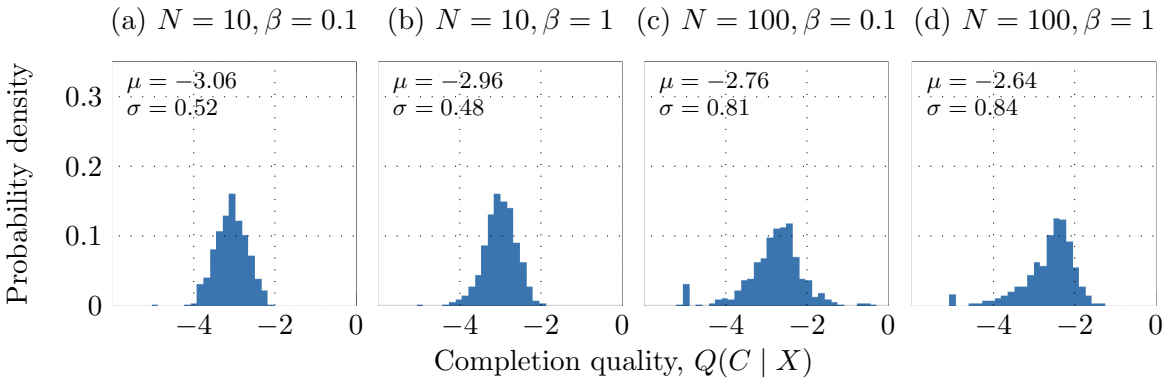


Figure 4.7: Probability distributions of text generated with biased TPS

4.2 Code Generation

The results of the HumanEval code generation benchmark are presented in Figure 4.8a–h, with the full results available in Table A.2 (see Appendix). We see that the biased TPS strategy outperforms all other strategies in zero-shot code performance, with a pass@1 score of 52.0% ($N = 100, \beta = 10$). This is followed by beam search with a pass@1 score of 51.2% ($B = 5$) and greedy search with a pass@1 score of 50.6%. Random sampling performs the worst, with a

pass@1 score of 39.9%.

There is also a consistent trend among strategies where increased diversity, as controlled by the strategies' parameters, leads to an increase in pass@10 score. This also appears to come with a decrease in pass@1 performance, indicative of a trade-off between exploration and exploitation. The trade-off is less pronounced in the biased TPS strategy with $N = 10$, where the pass@1 score increases to 48.7%, close to the performance of greedy search, despite a large increase in pass@10 score to 67.1%.

Interestingly, results did not significantly increase with the beam width B in beam search; in fact, performance decreased slightly with some values of B (e.g. $B = 3, 10$). This suggests the increased path exploration did not lead to superior code generation performance. Beam search also naturally had the lowest pass@10 scores as it only ever returns the top completion.

The sampling strategies (temperature, top-k, and top-p) also all show that diversity in completions only leads to an increase in pass@10 score to a certain point, beyond which the score begins to suffer, likely due to the scope of the search space becoming too large for viable solutions to be found.

4.3 Qualitative Analysis

Examples of text generated with all decoders can be found in the appendix. Refer to Figure A.31 for the input prompt used for these examples.

All strategies produced text that was grammatically correct and mostly coherent. However, the quality of the completions varied significantly between strategies. We see that the completion generated via greedy search (Figure 4.9) is highly repetitive from the start, repeating the phrase "they were very intelligent" multiple times. This shows the positive feedback loop from the model's own output, which is a common issue with greedy decoding. This is also somewhat present in the completions generated with beam search, particularly for lower beams, where the model is unable to explore the search space effectively (such as in Figure A.17).

The completions generated with sampling strategies are more diverse, with the random sampling completion (Figure 4.10) seeming much less focused on topic than the other completions. Top-k and top-p both do well to generate completions that are coherent and on-topic, with the top-p completion (Figure A.12) being particularly well-written.

Best-of-n completions do not appear to be significantly different from random sampling, often straying off-topic and lacking coherence. The biased TPS completions seem slightly more coherent, such as in Figure 4.11 ($N = 100, \beta = 1$), but still lacks the focus and quality of the completions generated with top-k and top-p sampling.

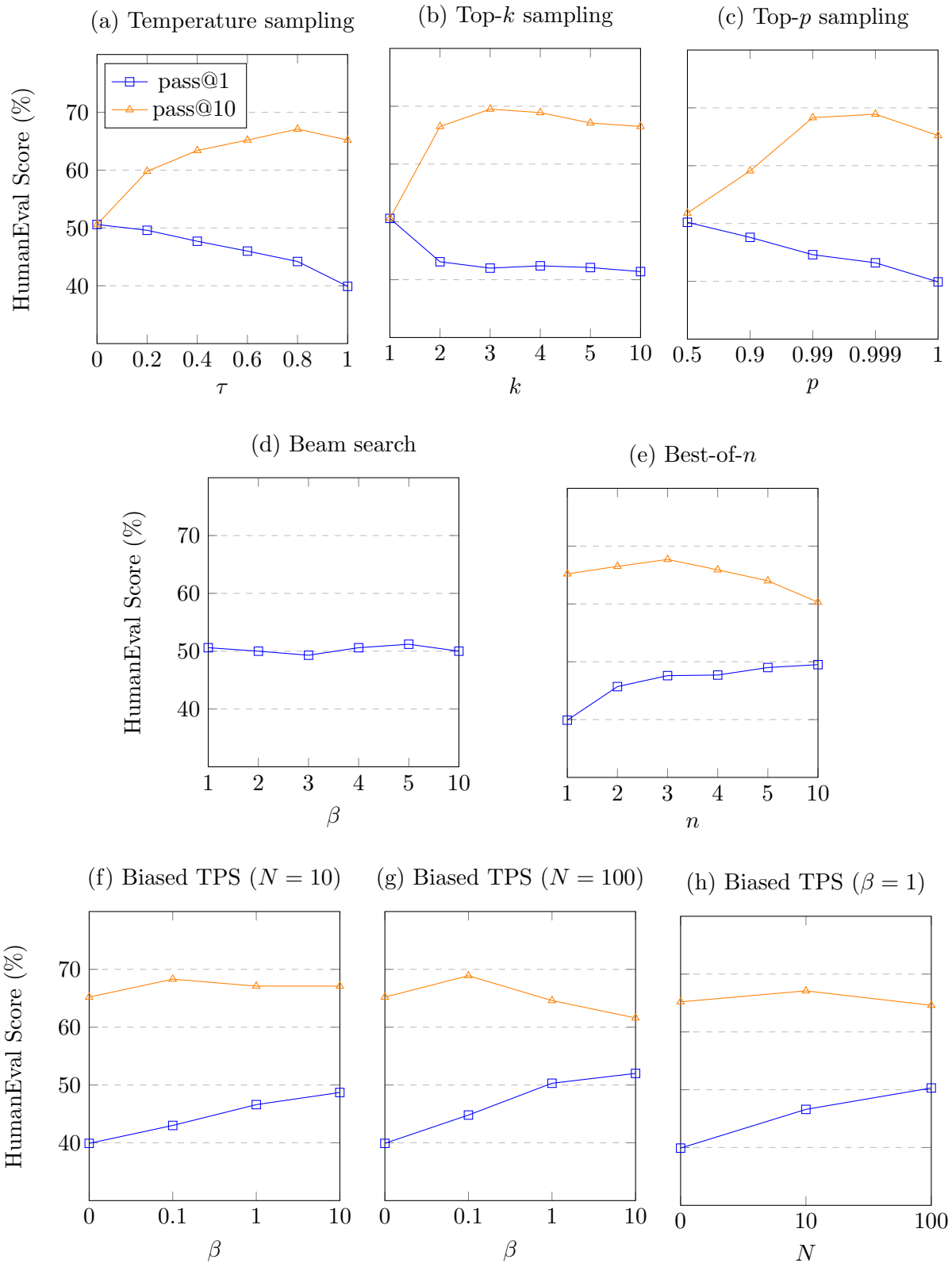


Figure 4.8: Human evaluation results for code generation. For $\tau = 0, k = 1, \beta = 1$ we use the results from greedy search. For $\tau = 1, p = 1, n = 1, \beta = 0, N = 0$ we use the results from random sampling.

```
"The unicorns were very intelligent, and they were very intelligent," said Dr.
David S. Siegel, a professor of anthropology at the University of California,
Berkeley. "They were very intelligent, and they were very intelligent, and
```

Figure 4.9: Text generation via greedy search showing repetition of phrases.

```
This adorable animal belongs to the "monstrous butstral tribe," or probably
unicorns. When exposed to the specific signals, some of the gender of the
animals you see in report reports occur close by, the researchers haven't
```

Figure 4.10: Random sampling completion showing off-topic content.

4.4 Further Analysis of Biased TPS Strategy

In addition to the previous results, we specifically analysed the impact of the β parameter, as well as the number of steps N, on the quality and diversity of completions. This is visualised in a heatmap in Figure 4.12.

Interestingly, we see that the quality of completions peaks at a particular value of $\beta \approx 0.15$ for all values of N. The same effect can be seen for negative values of β, where $\beta \approx -0.15$ produces the lowest quality completions. This suggests that specific values of β are more effective for finding high-quality completions. The number of steps N also has a significant impact on the quality of completions, with higher values of N generally producing better completions. Of course, values of β very close to 0.0 result in no biasing of the trajectory, and thus no improvement in quality.

We also analysed the impact of the number of steps N on the quality of completions, at different values of β. This is visualised in Figure 4.13. We see that the quality of completions continues towards the chosen bias as N increases at a rate roughly proportional to $\log N$. This suggests that the benefits of the strategy can be seen with relatively few steps, but the quality of completions continues to improve with more steps.

```
"This is so odd. Why would there be so little such diversity here on Earth?"
said Roberto E. Lattier, a biologist with the National Science Foundation in
Bilbao, Spain, who was not involved in the study.
```

Figure 4.11: Biased TPS Decoder, 100 steps, $\beta = 1$

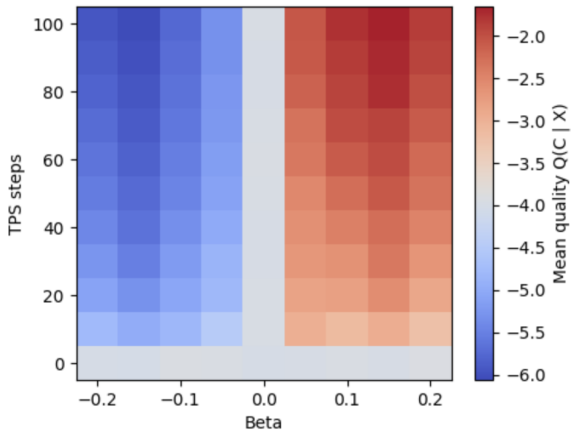


Figure 4.12: Heatmap of completion quality with biased TPS strategy, varying the bias parameter β and the number of steps N.

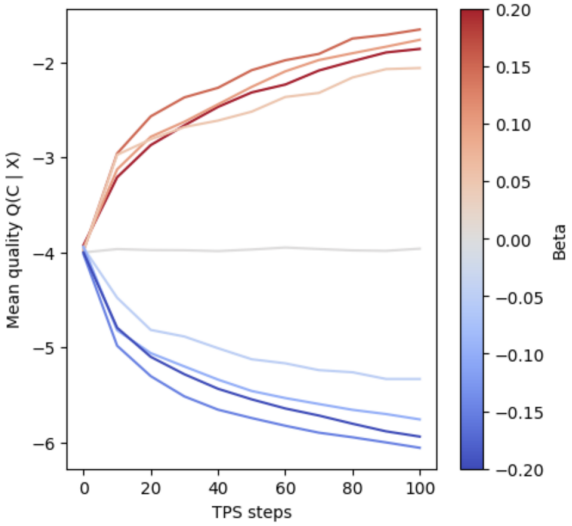


Figure 4.13: Impact of the number of steps N on the quality of completions with the biased TPS strategy, at different values of β.

5 Discussion and Conclusion

Our results demonstrate that decoding strategies are crucial in balancing the trade-off between text quality and diversity in neural text generation.

5.1 Analysis of Results

We find that the greedy search strategy performed well in terms of measurable quality metrics, with the second highest pass@1 score on the HumanEval benchmark, but largely suffered from its inability to explore the search space and navigate away from repeating words and phrases. The temperature, top-k, and top-p sampling strategies all offered nuanced control over this trade-off between quality and diversity. From our experiments, it would appear that temperature sampling with $\tau \approx 0.5$ held a nice balance between accuracy and diversity in both the code generation benchmark and qualitative analysis. Top-k had a significant impact on the quality of the generated code, even with the small values of k used, but did not allow for much diversity in open text generation (OTG). Top-p sampling showed a significant improvement in the quality of text generated in our qualitative analysis but did not make a substantial difference in the code generation benchmark. These findings indicate that code generation tasks require a more focused approach to decoding strategies, with fewer opportunities for exploration than OTG tasks.

The temperature sampling, top-k, and top-p strategies offered more nuanced control over this trade-off. We observed that lower temperature values and smaller k or p values tended to produce higher quality text, while higher values increased diversity at the expense of coherence. This aligns with previous findings in the literature [6, 36], and highlights the importance of careful parameter tuning for these strategies.

Beam search performed less well than anticipated, with a pass@1 score only marginally higher than greedy search. Certain beam widths, such as $B = 5$ and $B = 10$ actually performed worse than greedy search; this could be due to it favouring shorter sequences, which are perhaps less successful for coding tasks. It also naturally suffered from a lack of diversity, and the quality of the generated text was not significantly better than greedy search.

Best-of-n sampling showed promise for code generation once the number of samples was increased, but was not very effective for text generation. This suggests that the best-of-n strategy may be more effective in tasks where the search space is more constrained (like code generation). The standard deviation of response qualities also increased, perhaps indicating that the strategy was not always able to find a better completion over random sampling.

5.2 Evaluation of Biased TPS Strategy

Our novel biased TPS strategy showed promising results in terms of both quality and diversity. While the sampling strategies we explored in the previous section offered a trade-off between quality and diversity, the biased TPS strategy showed improvements in both aspects. The code generation tasks in particular benefited from this strategy, with the highest pass@1 scores achieved across all strategies and respectable pass@10 scores. This is despite the relatively low number of steps used ($N = 100$), and the graph in Figure 4.8 shows a clear upward trend in quality as the number of steps increases.

It is also interesting to note the impact of the β parameter on the generated text. The results in Figure A.1 show little difference between $\beta = 1$ and $\beta = 10$ for the quality of OTG, but a fairly significant difference for code generation. This suggests that the β parameter may have a more pronounced effect on tasks with a constrained search space, and that the optimal value of β may vary depending on the task.

The further analysis in Figure 4.12 shows that an optimal value of β exists, at least for that specific OTG prompt, which leads to the highest results. This can be explained by the fact that the β parameter controls the balance between exploration and exploitation. As values of β closer to 0 encourage more exploration and less bias towards the best samples, it is possible that there is an optimal value of β to maximise quality through a balance of exploration and exploitation, which may be different for different tasks, prompts, or completion lengths.

5.3 Ethical Considerations

There is, of course, a discussion to be had about the ethical implications of the work presented in this project. The biased TPS strategy, while showing promising results in terms of quality and diversity, also raises concerns about the potential for amplifying biases in the generated text. The biased TPS strategy may exacerbate biases present in the training data, as it favours samples that are more representative of the training data, potentially leading to biased or discriminatory text that would not arise from other decoding strategies. Further, the ability to bias text generation to any objective function (which our strategy enables) could potentially be misused for generating misinformation or harmful material.

5.4 Limitations and Challenges Faced

Several limitations of this project should be acknowledged:

- **Computational Constraints**: Due to resource limitations, we were only able to run the TPS algorithm for a maximum of 100 iterations. Further exploration with higher iteration counts may yield additional insights. It also may have been beneficial to experiment with higher values of B and n with beam search and best-of-n respectively. More challenging benchmarks which require longer generations or more powerful models may also have been

beneficial.

- **Model Size**: Our experiments were run with GPT-2 and Phi-1 for the OTG and code generation tasks respectively, which are weak models compared to the SOTA language models that are currently available. While suitable for comparative analysis, the absolute performance and potentially the relative effectiveness of different strategies may vary with larger, more capable models.
- **Quality Metric**: While our quality metric (defined in Section 3.1.1) provides some valuable insights, it certainly does not fully capture all aspects of text quality. It is also limited in nature by the power of the model itself. It provides more insight into the pathfinding abilities of the decoding strategies rather than the quality of the text itself.
- **Limited Task Diversity**: Our evaluation focused primarily on open-ended text generation and code completion. Analysis into the effects of this strategy on other tasks, such as logical reasoning, or other metrics, such as the automated readability index (ARI), would be beneficial.

  A key challenge of this project was optimising our utilisation of the high-performance capabilities and enabling batched implementations for global strategies.

5.5 Project Implications and Future Work

We've shown with this work that transition path sampling – a method primarily used in physical simulations and molecular dynamics – can be utilised to leverage the power of language models. The effectiveness of this approach suggests that there may be untapped potential in applying techniques from other domains to neural text generation. We've also shown both qualitatively and quantitatively the effects that different decoding strategies can have on the output and performance of language models and highlighted the need for nuanced evaluation metrics to accurately capture the many dimensions of text evaluation. Future work could explore:

- Extending the biased TPS strategy to allow for additional parameters (e.g. temperature or top-k);
- Examining the performance of this strategy (as well as other global strategies) on more demanding tasks, such as complex chain-of-thought logical reasoning;
- Exploring the performance scaling of the strategy on larger models, and with a wider range of values for N and β;
- Investigating the TPS strategy in the context of metrics other than probability, i.e. inducing particular characteristics like politeness;
- Developing more comprehensive evaluation frameworks that incorporate human judgements, or evaluation from larger models such as GPT-4;
- Using the strategy to generate high-quality synthetic datasets, e.g. for knowledge

distillation.

5.6 Conclusion

In conclusion, this study has demonstrated the significant impact that decoding strategies can have on the output of language models. Our novel biased TPS approach shows promise in balancing quality and diversity, particularly in structured tasks like code generation. While challenges remain in terms of computational efficiency and comprehensive evaluation, this work lays a foundation for further research into optimising the decoding process for neural text generation.

As language models continue to advance in capability and scale, the importance of sophisticated decoding strategies will only grow. By carefully considering the trade-offs between quality, diversity, and task-specific performance, we can develop more effective and responsible approaches to leveraging these powerful tools across a wide range of applications.

References

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

[2] Gemini Team, Rohan Anil, Sebastian Borgeaud et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.

[3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan et al. *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. 2023. arXiv: 2303.12712 [cs.CL]. URL: https://arxiv.org/abs/2303.12712.

[4] Cameron R. Jones and Benjamin K. Bergen. *People cannot distinguish GPT-4 from a human in a Turing test*. 2024. arXiv: 2405.08007 [cs.HC]. URL: https://arxiv.org/abs/2405.08007.

[5] Matthew Renze and Erhan Guven. *The Effect of Sampling Temperature on Problem Solving in Large Language Models*. 2024. arXiv: 2402.05201 [cs.CL]. URL: https://arxiv.org/abs/2402.05201.

[6] Ari Holtzman et al. *The Curious Case of Neural Text Degeneration*. 2020. arXiv: 1904.09751 [cs.CL]. URL: https://arxiv.org/abs/1904.09751.

[7] Hugo Touvron, Louis Martin, Kevin Stone et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[8] Christoph Dellago, Peter G. Bolhuis and David Chandler. "Efficient transition path sampling: Application to Lennard-Jones cluster rearrangements". In: *The Journal of Chemical Physics* 108.22 (June 1998), pp. 9236–9245. ISSN: 0021-9606. DOI: 10.1063/1.476378. eprint: https://pubs.aip.org/aip/jcp/article-pdf/108/22/9236/19055790/9236_1_online.pdf. URL: https://doi.org/10.1063/1.476378.

[9] "The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954". In: 2006. URL: https://api.semanticscholar.org/CorpusID:132677.

[10] Joseph Weizenbaum. *Computer Power and Human Reason: From Judgment to Calculation*. San Francisco: W. H. Freeman, 1976.

[11] Güven Güzeldere and Stefano Franchi. "Dialogues with colorful personalities of early AI". In: *Stanford Humanities Review* 4.2 (July 1995). Retrieved from https://web.archive.org/web/20110425191843/http://www.stanford.edu/group/SHR/4-2/text/dialogues.html. URL: https://web.archive.org/web/20110425191843/http://www.stanford.edu/group/SHR/4-2/text/dialogues.html.

[12] Peter F. Brown, Vincent J. Della Pietra et al. "Class-Based n-gram Models of Natural Language". In: *Computational Linguistics* 18.4 (1992), pp. 467–480. URL: https://aclanthology.org/J92-4003.

[13] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of Neural Information Processing Systems (NIPS)*. 2012, pp. 1106–1114.

[14] Ilya Sutskever, James Martens and Geoffrey Hinton. "Generating text with recurrent neural networks". In: ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 1017–1024. ISBN: 9781450306195.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017), pp. 5998–6008.

[16] OpenAI. *GPT-3.5 Tokenizer*. Accessed: 16/08/2024. 2023. URL: https://platform.openai.com/tokenizer.

[17] Zihao Li et al. *Quantifying Multilingual Performance of Large Language Models Across Languages*. 2024. arXiv: 2404.11553 [cs.CL]. URL: https://arxiv.org/abs/2404.11553.

[18] Nicolo Micheletti et al. *Exploration of Masked and Causal Language Modelling for Text Generation*. 2024. arXiv: 2405.12630 [cs.CL]. URL: https://arxiv.org/abs/2405.12630.

[19] Edward Y. Chang. "Examining GPT-4: Capabilities, Implications and Future Directions". In: (2023). Accessed: [05/09/2024]. URL: https://www.researchgate.net/profile/Edward-Chang-22/publication/374753069_Examining_GPT-4's_Capabilities_and_Enhancement_with_SocraSynth/links/6561a58ece88b870310e60ef/Examining-GPT-4s-Capabilities-and-Enhancement-with-SocraSynth.pdf.

[20] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[21] Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[22] Amirhossein Kazemnejad et al. *The Impact of Positional Encoding on Length Generalization in Transformers*. 2023. arXiv: 2305.19466 [cs.CL]. URL: https://arxiv.org/abs/2305.19466.

[23] Kevin Meng et al. *Locating and Editing Factual Associations in GPT*. 2023. arXiv: 2202.05262 [cs.CL]. URL: https://arxiv.org/abs/2202.05262.

[24] Brandon McKinzie et al. *MM1: Methods, Analysis & Insights from Multimodal LLM Pre-training*. 2024. arXiv: 2403.09611 [cs.CV]. URL: https://arxiv.org/abs/2403.09611.

[25] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[26] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[27] Suriya Gunasekar et al. *Textbooks Are All You Need*. 2023. arXiv: 2306.11644 [cs.CL]. URL: https://arxiv.org/abs/2306.11644.

[28] Yukun Zhu et al. "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books". In: *arXiv preprint arXiv:1506.06724*. 2015.

[29] Will Knight. "OpenAI's CEO Says the Age of Giant AI Models Is Already Over. Sam Altman says the research strategy that birthed ChatGPT is played out and future strides in artificial intelligence will require new ideas". In: *Wired* (Apr. 2023). Accessed: 04/09/2024. URL: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/.

[30] Ahmad Faiz et al. *LLMCarbon: Modeling the end-to-end Carbon Footprint of Large Language Models*. 2024. arXiv: 2309.14393 [cs.CL]. URL: https://arxiv.org/abs/2309.14393.

[31] Jesse Dodge et al. *Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping*. 2020. arXiv: 2002.06305 [cs.CL]. URL: https://arxiv.org/abs/2002.06305.

[32] Yann LeCun. *A Path Towards Autonomous Machine Intelligence*. Tech. rep. Version 0.9.2. Accessed: 28/08/2024. New York University and Meta AI Research, June 2022. URL: https://openreview.net/pdf?id=BZ5a1r-kVsf.

[33] Rylan Schaeffer, Brando Miranda and Sanmi Koyejo. *Are Emergent Abilities of Large Language Models a Mirage?* 2023. arXiv: 2304.15004 [cs.AI]. URL: https://arxiv.org/abs/2304.15004.

[34] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: 2311.05232 [cs.CL]. URL: https://arxiv.org/abs/2311.05232.

[35] Yuu Jinnai et al. *Regularized Best-of-N Sampling to Mitigate Reward Hacking for Language Model Alignment*. 2024. arXiv: 2404.01054 [cs.CL]. URL: https://arxiv.org/abs/2404.01054.

[36] Matthew Renze and Erhan Guven. *The Effect of Sampling Temperature on Problem Solving in Large Language Models*. 2024. arXiv: 2402.05201 [cs.CL]. URL: https://arxiv.org/abs/2402.05201.

[37] Yilin Yang, Liang Huang and Mingbo Ma. *Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation*. 2018. arXiv: 1808.09582 [cs.CL]. URL: https://arxiv.org/abs/1808.09582.

[38] Dominic C Rose, Jamie F Mair and Juan P Garrahan. "A reinforcement learning approach to rare trajectory sampling". In: *New Journal of Physics* 23.1 (Jan. 2021), p. 013013. DOI: 10.1088/1367-2630/abd7bd. URL: https://dx.doi.org/10.1088/1367-2630/abd7bd.

[39] Jamie F. Mair, Dominic C. Rose and Juan P. Garrahan. *Training neural network ensembles via trajectory sampling*. 2023. arXiv: 2209.11116 [cond-mat.stat-mech]. URL: https://arxiv.org/abs/2209.11116.

[40] F. Jelinek et al. "Perplexity—a measure of the difficulty of speech recognition tasks". In: *The Journal of the Acoustical Society of America* 62.S1 (Aug. 2005), S63–S63. ISSN: 0001-4966. DOI: 10.1121/1.2016299. eprint: https://pubs.aip.org/asa/jasa/article-pdf/62/S1/S63/11558910/s63_5_online.pdf. URL: https://doi.org/10.1121/1.2016299.

[41] Alessio Miaschi et al. “What Makes My Model Perplexed? A Linguistic Investigation on Neural Language Models Perplexity”. In: *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Ed. by Eneko Agirre, Marianna Apidianaki and Ivan Vulić. Online: Association for Computational Linguistics, June 2021, pp. 40–47. DOI: 10.18653/v1/2021.deelio-1.5. URL: https://aclanthology.org/2021.deelio-1.5.

[42] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[43] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. 2021. arXiv: 2009.03300 [cs.CY]. URL: https://arxiv.org/abs/2009.03300.

[44] Krishna Pillutla et al. *MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers*. 2021. arXiv: 2102.01454 [cs.CL]. URL: https://arxiv.org/abs/2102.01454.

[45] Vilém Zouhar et al. “A Formal Perspective on Byte-Pair Encoding”. In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 598–614. DOI: 10.18653/v1/2023.findings-acl.38. URL: https://aclanthology.org/2023.findings-acl.38.

A Appendix

A.1 Algorithms

Algorithm A.1 Byte-Pair Encoding [45]

Input: $D \leftarrow$ Dataset, e.g. a text corpus
Input: $N \leftarrow$ Maximum vocabulary size
Output: Vocabulary vector V
  $V \leftarrow$ {Initialise vocabulary as all single characters found in D}
  while Length of $V < N$ **do**
    $v_a, v_b \leftarrow$ Find the two tokens most often adjacent in D
    $v_k \leftarrow$ Merge v_i, v_j into a new, single token
  end while

Algorithm A.2 Text Generation with Probability Calculations for Local Strategies

Input: Input sequence, $X = [x_0, x_1, \ldots, x_n]$
Input: Maximum number of tokens to generate, i
Output: Extended sequence X
Output: Joint log probability of sequence, p
  $p \leftarrow 0$ ▷ Input text has probability of 100%
  for range $1 \rightarrow i$ **do**:
    if $V_{\text{EOS}} \in X$ **then** ▷ Check if end token is in sequence
      break
    end if
    $Y \leftarrow$ Output logits from model for sequence X
    $t \leftarrow$ Chosen token from decoding strategy with input Y
    Append t to the end of X
    $p \leftarrow p + \log \sigma(Y)_t$ ▷ Add log softmax of chosen token
  end for

A.2 Probability Statistics and Trajectory Qualities

Table A.1: Quality statistics for all tested decoding strategies. All text for this experiment was generated with GPT-2 [1]. For full details behind these results, refer to Section 3.3.1.

Decoding Strategy	**Parameter**	$Q(C \mid X)$	
		Mean	**Std. Dev.**
Greedy search	—	-1.46	0.00
Random sampling	—	-3.63	0.57
Temperature sampling	$\tau = 0.2$	-1.41	0.16
	$\tau = 0.4$	-1.50	0.18
	$\tau = 0.6$	-1.77	0.23
	$\tau = 0.8$	-2.42	0.38
Top-k sampling	$k = 2$	-1.48	0.17
	$k = 3$	-1.61	0.19
	$k = 4$	-1.69	0.19
	$k = 5$	-1.75	0.20
	$k = 10$	-1.96	0.23
Top-p sampling	$p = 0.5$	-1.71	0.22
	$p = 0.9$	-2.94	0.45
	$p = 0.99$	-3.51	0.54
	$p = 0.999$	-3.62	0.54
Beam search	$\beta = 2$	-1.41	0.00
	$\beta = 3$	-0.88	0.00
	$\beta = 4$	-0.88	0.00
	$\beta = 5$	-0.91	0.00
	$\beta = 10$	-0.82	0.00
Best-of-n	$n = 2$	-3.38	0.49
	$n = 3$	-3.22	0.51
	$n = 4$	-3.16	0.54
	$n = 5$	-3.13	0.58
	$n = 10$	-3.06	0.74
Shooting TPS	$N = 10, \beta = 0.1$	-3.06	0.52
	$N = 10, \beta = 1$	-2.96	0.48
	$N = 10, \beta = 10$	-2.97	0.50
	$N = 100, \beta = 0.1$	-2.76	0.81
	$N = 100, \beta = 1$	-2.64	0.84
	$N = 100, \beta = 10$	-2.65	0.85

A.3 HumanEval Benchmark Results

Table A.2: Full results of HumanEval benchmark under various decoding strategies. All tests were performed with the Phi-1 language model [27].

Decoding Strategy	**Parameter**	**HumanEval Score**	
		pass@1	**pass@10**
Greedy search	—	50.6%	–
Random sampling	—	39.9%	65.2%
Temperature sampling	$\tau = 0.2$	49.6%	59.8%
	$\tau = 0.4$	47.7%	63.4%
	$\tau = 0.6$	46.0%	65.2%
	$\tau = 0.8$	44.2%	67.1%
Top-k sampling	$k = 2$	43.1%	66.5%
	$k = 3$	42.0%	69.5%
	$k = 4$	42.4%	68.9%
	$k = 5$	42.1%	67.1%
	$k = 10$	41.4%	66.5%
Top-p sampling	$p = 0.5$	50.2%	51.8%
	$p = 0.9$	47.6%	59.1%
	$p = 0.99$	44.6%	68.3%
	$p = 0.999$	43.2%	68.9%
Beam search	$B = 2$	50.0%	–
	$B = 3$	49.3%	–
	$B = 4$	50.6%	–
	$B = 5$	51.2%	–
	$B = 10$	50.0%	—
Best-of-n	$n = 2$	45.7%	66.5%
	$n = 3$	47.6%	67.5%
	$n = 4$	47.7%	65.9%
	$n = 5$	49.0%	64.0%
	$n = 10$	49.5%	60.3%
Biased TPS	$N = 10, \beta = 0.1$	43.0%	68.3%
	$N = 10, \beta = 1$	46.6%	67.1%
	$N = 10, \beta = 10$	48.7%	67.1%
	$N = 100, \beta = 0.1$	44.8%	68.9%
	$N = 100, \beta = 1$	50.3%	64.6%
	$N = 100, \beta = 10$	52.0%	61.6%

A.4 Generated Text Examples

```
"The unicorns were very intelligent, and they were very intelligent," said Dr.
David S. Siegel, a professor of anthropology at the University of California,
Berkeley. "They were very intelligent, and they were very intelligent, and
```

Figure A.1: Greedy search

```
This adorable animal belongs to the "monstrous butstral tribe," or probably
unicorns. When exposed to the specific signals, some of the gender of the
animals you see in report reports occur close by, the researchers haven't
```

Figure A.2: Random sampling

```
"We've never seen a unicorn in the wild before," said Dr. David S. Karp, a
professor of anthropology at the University of California, Santa Barbara.
"We've never seen a unicorn in the wild before. We've
```

Figure A.3: Temperature sampling, $\tau = 0.2$

```
The researchers, who are from the University of California, San Diego, and the
University of California, San Francisco, analyzed the genomes of more than 1,000
young male and female unicorns in the wild. They found that the young males
```

Figure A.4: Temperature sampling, $\tau = 0.4$

```
The female unicorn, named Anacor, was found in a cave in the Andes Mountains,
located in a remote region of the Andes Mountains. (Photo: Wikimedia Commons)
The researchers say this discovery is the first
```

Figure A.5: Temperature sampling, $\tau = 0.6$

```
DeLucaño, who became a national expert on unicorns because of close studies of
them, pledged to find out the truth. "The other day I was very shocked,"
DeLucaño said. The researchers presented their
```

Figure A.6: Temperature sampling, $\tau = 0.8$

```
The researchers found that when the unicorns were in the valley, they were able
to communicate with each other in a way that was similar in language to that
found on the ground. The unicorns were able to communicate with one another in
```

Figure A.7: Top-k sampling, $k = 2$

```
The unicorns were so well-behaved that they were very happy and happy with their
life," said Dr. Michael Siegel, a professor of anthropology and director of the
Institute for Animal Behavior Research. "They were very well-
```

Figure A.8: Top-k sampling, $k = 3$

```
The researchers were able to decipher the words that the unicorn used to
communicate with the animals, and they found a surprising amount of information
about the animals in their language. "The unicorns were able to communicate
```

Figure A.9: Top-k sampling, $k = 4$

```
"The unicorns were so well spoken, they didn't even need to be spoken," said
study researcher, Prof. Yves Le Couteur-Gagnon. The study's lead author, Prof.
Yves Le
```

Figure A.10: Top-k sampling, $k = 5$

```
"It sounds like the English language is a lot more natural, and that's a shame,"
says Dr. John B. Clements, assistant professor at the University of Colorado.
"The unicorns are so unique." The study was published
```

Figure A.11: Top-k sampling, $k = 10$

```
The research was published in the journal Proceedings of the National Academy
of Sciences. The unicorns were found in the area by a team led by Marcel
Dutroux, a researcher at the Université de Lausanne in Switzerland
```

Figure A.12: Top-p sampling, $p = 0.5$

```
Yet the unicorns did not speak English, either. At least, not what they used
to. "It is quite natural for a unicorn to speak English well," says Parvati
Verma, Ph.D., lead author of the paper.
```

Figure A.13: Top-p sampling, $p = 0.9$

```
So the Himalayan herd was in the midst of an extinction. In a daring research
by astrophysicist Neil Carrier, he found out that the all-beloved Fusiliers
of the Himalayas, disappearing from inaccessible places like the
```

Figure A.14: Top-p sampling, $p = 0.99$

```
Mark Cussen and Stuart Beauregardt, both with Duke University in the US, were
among the first to have specialized in subtler genetic analysis to follow
these animals. They call it workarounds. If the genomes of the
```

Figure A.15: Top-p sampling, $p = 0.999$

```
"The unicorns are very intelligent, and they have a very strong sense of
smell," said lead author Dr. David S. Siegel of the University of California,
San Diego. The study was published in the journal Nature Communications
```

Figure A.16: Beam search, $B = 2$

```
The researchers found that the unicorns were able to communicate with each
other in a way that was similar to that of humans. "The unicorns were able
to communicate with each other in a way that was similar to that of humans
```

Figure A.17: Beam search, $B = 3$

```
According to the researchers, the unicorns were able to communicate with each
other in a way that was similar to that of humans. "The unicorns were able
to communicate with each other in a way that was similar to that of
```

Figure A.18: Beam search, $B = 4$

```
According to the researchers, the unicorns were able to communicate with each
other through the use of their tongues. "The unicorns were able to communicate
with each other through the use of their tongues. The unicorns were able to
```

Figure A.19: Beam search, $B = 5$

```
The study, published in the journal Proceedings of the National Academy of
Sciences, is the first to show that unicorns are able to communicate with
humans. "This is the first time that unicorns have been able to communicate with
```

Figure A.20: Beam search, $B = 10$

```
We were inclined to believe that it was rather probabilistic, so we decided
to stick to that laws because we didn't know what to expect," says Charles
Nisbet, an ecologist from the UCLA Extension's Southwest Research Institute
```

Figure A.21: Best-of-n, $n = 2$

```
A large 20 kilograms wide male male lived in this isolated valley between 18
and 24 meters in circumference. The area was separated by his main land bridge
which was a long tunnel deemed as the longest in the world. The journey was 100
```

Figure A.22: Best-of-n, $n = 3$

```
Teens and teens were introduced to English at a young age, but unicorns evolved
quickly, Plabstes said. "This translates to a worldwide goal-based approach that
accelerates the development of human language development and suggests an
```

Figure A.23: Best-of-n, $n = 4$

```
In one common word, unicorns were known as "briggity boys," though most of
the recordings were in English. Civic violas and goats sometimes played flat
notes in the room. The pair of unicorns had begun making new tongues
```

Figure A.24: Best-of-n, $n = 5$

```
Researchers from Denmark discovered that during this time period, unicorns soon
used to call their own name rather than the most common name of their nest.
"Bulgaria was a really rough place to live because of the poor lifestyle of
```

Figure A.25: Best-of-n, $n = 10$

"They could complete the thing through natural means" by mingling similar English words, said Dr. Feskan Topmisoglu, the conservatory's director. It turned out that the unicorns spoke perfect English and were fluent Indonesian.

Figure A.26: Biased TPS Decoder, 10 steps, $\beta = 0.1$

This discovery, according to the journal Science, supports longstanding animal-animal hybridism amongst native non-extinct animals, which in the wild are more closely related to native humans.

Figure A.27: Biased TPS Decoder, 10 steps, $\beta = 1$

The Americans, in the beginning, were said to be eager to help convert these songs into English idioms. The discovery invigorated exploration of just how essentially modern the languages are, and how much they can be used for, to

Figure A.28: Biased TPS Decoder, 10 steps, $\beta = 10$

However, they did not speak speakers of the non-Caucasian languages of South America. Email

Figure A.29: Biased TPS Decoder, 100 steps, $\beta = 0.1$

"This is so odd. Why would there be so little such diversity here on Earth?" said Roberto E. Lattier, a biologist with the National Science Foundation in Bilbao, Spain, who was not involved in the study.

Figure A.30: Biased TPS Decoder, 100 steps, $\beta = 1$

"Most of them weren't even inhabitants of Europe...But they all had high birthrates that were too low to survive in the wild," says Clarice Wilson, a geneticist at the National Center for Missing and Exploited Children in

Figure A.31: Biased TPS Decoder, 100 steps, $\beta = 10$