HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTER THESIS

Mr.
Max Schlosser, B.Sc.

Development of a semi-automated process for prompt generation to improve readability in the context of text generation

Mittweida, September 2024

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Development of a semi-automated process for prompt generation to improve readability in the context of text generation

Author:
Max Schlosser

Course of Study:
Media Informatics and Interactive Entertainment

Seminar Group:
MI22w1-M

First Examiner:
Prof. Dr.-Ing. Christian Roschke

Second Examiner:
Manuel Heinzig M.Sc.

Submission:
Mittweida, 05.09.2024

Defense/Evaluation:
Mittweida, 2024

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Entwicklung eines semi-automatisierten Verfahrens zur Promptgenerierung für die Verbesserung der Lesbarkeit im Kontext der Textgenerierung

Autor:
Max Schlosser

Studiengang:
Medieninformatik und Interaktives Entertainment

Seminargruppe:
MI22w1-M

Erstprüfer:
Prof. Dr.-Ing. Christian Roschke

Zweitprüfer:
Manuel Heinzig M.Sc.

Einreichung:
Mittweida, 05.09.2024

Verteidigung/Bewertung:
Mittweida, 2024

Bibliographic Description

Schlosser, Max:
Development of a semi-automated process for prompt generation to improve readability in the context of text generation. – 2024. – 73 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences, Master Thesis, 2024.

Abstract

This thesis introduces a semi-automated process for optimizing prompt generation using Reinforcement Learning to improve text readability of content generated by Large Language Models. A novel readability metric normalization technique is employed to ensure consistent evaluation across text samples. The research utilizes a distributed system architecture to integrate multiple services, including word pool scraping, prompt generation and text evaluation, enabling scalable and efficient training of an agent. Results indicate significant improvements in text readability, demonstrating the effectiveness of the proposed approach.

Contents

List of Figures

List of Tables

Acronyms

AGI ..... Artificial General Intelligence

AI ..... Artificial Intelligence

BBoM ..... Big Ball of Mud

DBMS ..... Database Managament System

DQN ..... Deep Q Network

GPT ..... Generative Pretrained Transformer

HTTP ..... Hypertext Transfer Protocol

JSON ..... JavaScript Object Notation

LLM ..... Large Language Model

MDP ..... Markov Decision Process

ML ..... Machine Learning

NLG ..... Natural Language Generation

NLP ..... Natural Language Processing

NLU ..... Natural Language Understanding

ORM ..... Object-Relational Mapping

POS ..... Parts of Speech

REST ..... Representational State Transfer

RL ..... Reinforcement Learning

RLHF ..... Reinforcement Learning from Human Feedback

TDL ..... Temporal Difference Learning

URI ..... Uniform Resource Identifier

1 Introduction

Advances in Artificial Intelligence (AI) in recent years have been nothing short of a digital revolution. While significant progress has already been made in the scientific context for a long span of time, the public attention to the potential, applicability, and also threats of the field has mostly been drawn through *language models* [1, pp. 57-58] [2]. The ability to engage in simple conversations or delegate complex tasks to human-like chatbots is inspiring and helpful for many people and professional applications [3]. Furthermore, the integration of AI into various sectors has led to transformative changes, influencing industries such as healthcare, finance, education, and entertainment [4] [5].

Language models, particularly the Generative Pretrained Transformer (GPT) models provided by OpenAI[1], have been at the forefront of this revolution. These models are not only used by millions of private users daily but are also integrated by many economic actors into existing or entirely new applications, ranging from automated customer support to content generation [6] [7] [8]. For example, language models have been employed in generating technical documentation, creative writing, and even in assisting programming tasks by suggesting code snippets [9] [10]. This widespread adoption underscores the versatile nature of language models and their potential to revolutionize numerous fields.

The content, structure, and coherence of a text generated by a Large Language Model (LLM) are crucially dependent on the input provided to the model (also defined as the *prompt*), making targeted construction of this input (also called *prompt engineering*) increasingly important for reliable results [11]. Effective design of prompts is essential not only for generating coherent and relevant text, but also for guiding the model towards producing content that aligns with specific requirements or constraints [12]. As the use of LLMs expands, the field of prompt engineering has emerged as a critical area of research and development, focusing on optimizing the interaction between the user and the model to enhance the quality of generated outputs [13].

As manual prompt design does often need many adjustments before desired results are achieved, automatic prompt construction methods have gained popularity and showed promising results [14] [15]. Combining targeted methods of prompt engineering with automatization leads to *semi-automatic* approaches.

This thesis aims to optimize text readability through semi-automatically designed prompts, utilizing algorithms of Reinforcement Learning (RL). RL has proven to be a successful and flexible approach for many problem classes associated with dynamic environments [16, p. 3]. By applying RL to the task of prompt optimization, this work seeks to explore whether improving the readability of text, based on objective factors, is possible using a LLM generating text samples. Previous studies have demonstrated the potential of RL in optimizing various aspects of Machine Learning (ML) models, including LLMs, by refining parameters and adjusting strategies to maximize performance [17] [18].

[1] https://openai.com/

In addition to the introduction, this work is structured into five chapters. Chapter 2 introduces the fundamental theoretical concepts and various approaches associated with the subject. Chapter 3 outlines the methodology applied to examine the use of RL to improve the readability of content generated by text generative AI. A preliminary experiment is conducted to test basic assumptions of the methodology. The implementation, extending various aspects of the preliminaries, follows in Chapter 4. Various software components are distributed efficiently to train the main system used to automatically fill prompt input for text generative systems. Chapter 6 discusses the outcomes of the study by interpreting the derived data. The discussion also considers how future research could be carried out building on this work to enhance results. The last Chapter 7 reflects on this work by concluding the most important findings.

2 Theoretical Background

This chapter will introduce all the important theoretical concepts for the following chapters.

2.1 Large Language Models

Language models are part of Natural Language Processing (NLP), which counts as a separate branch of AI. The main aim of NLP is to provide a link between human language and its understanding by computers, therefore empowering systems to "process, analyze and respond to natural language data" [19, p. 9].

NLP is divided into two sub-processes: [20, pp. 95-96]

- **Natural Language Understanding (NLU)**: The system is able to understand the human language by interpreting the structure, components, and meaning of sentences.
- **Natural Language Generation (NLG):** The system generates novel text content. This step requires NLU beforehand.

A LLM is generally defined as a NLG-providing language model that "has a large amount of parameters and is trained on a massive dataset" [19, p. 82]. Over the years, various major technical advancements have led to significant improvements of different aspects of NLP and especially text generation.

This work will take into consideration *conversational models* which provide a chat-like environment, enabling users to chat with a LLM like with another human. With the raising popularity of the service ChatGPT[2] created by OpenAI, other providers like Google or Meta started working on their own models supporting conversations, making this particular type of model the most user-friendly option. [19, p. 2]

2.1.1 Prompt Engineering

The term *prompt engineering* refers to "the art and science of crafting effective input prompts to guide the behavior of LLM" [19, p. 109].

As generating text content for the experiments in this work will be based on a mix of manual constructed sentence structures and automatically generated passages, a generic approach for designing prompts leading to better text quality will be used. The applied framework known as *CLEAR* follows five core principles: [21]

- **Concise:** Prompts should be as short and precise as possible. Therefore, unnecessary words or information should be avoided.
- **Logical:** Generation results are improved by providing a logical flow to the prompt. A good way to do this is specifying a start and end aspect of the content to be generated.

[2] https://chatgpt.com/

- **Explicit:** By explicitly providing information on output format and scope, LLMs are less likely to provide irrelevant information.
- **Adaptive:** When faced with unsatisfying results, prompt designers need to be flexible in adapting their input prompts. This may mean specializing or limiting the content.
- **Reflective:** The generated content should be critically reflected upon to evaluate results. This aspect, which is urgently given the error-prone nature of generative AI, can be incorporated into the revision of prompts.

2.2 Linguistics

This section will deal with linguistic foundations, which are substantial for using LLMs' language inference and designing corresponding prompts. Therefore, various language rudiments and their usage for NLP and NLU as well as prompt design will be taken into consideration.

2.2.1 Basics

Linguistics is generally defined as "the scientific study of language or of particular languages". The field of research is broadly divided into five subfields: **phonetics** and **phonology** deal with the speech sounds and sound systems of various languages.
Due to this work's limitations, only written English language is taken into account, excluding phonetics and phonology as points of interest for spoken language. The most important principles for the other branches, **morphology**, **syntax** and **semantics**, will be examined in the following. [22, p. 2-3]

The term *morphology* refers to the "internal structure of words" as well as the possible mutations of word components allowing for new vocabulary combinations.
Exchangeable parts, so-called *morphemes*, can be clustered using three main criteria. The grade of **autonomy** determines whether a morpheme can occur solely ("free") or is always bound to at least one other morpheme to form a meaningful word. **Function and meaning** of a morpheme describe whether "lexical or grammatical information" are conveyed. While bound morphemes are distinguished between *derivational* (creating new lexical meanings) and *inflectional* (grammatical alternations), free morphemes are divided into *lexical word classes* building the semantic base and *grammatical/function word classes* conveying grammatical meaning, e.g., in the form of articles or pronouns. Lastly, the **position** specifies where bound morphemes are injected to modify a word. All types of morphemes according to the definitions provided are visually summarized in Figure 2.1. [23, p. 51-52]

Syntactical linguistic components specify the ruleset used to combine smaller components into grammatical correct ones. Therefore, lexemes are classified into various *Parts of Speechs (POSs)* such as nouns, verbs or prepositions. Characteristics of word classes may vary, making different mutations of a word follow heterogeneous rules. This *prototypicality* determines, for example, whether comparative and superlative forms of an adjective are formed using a specific rule. [23, pp. 86-88]

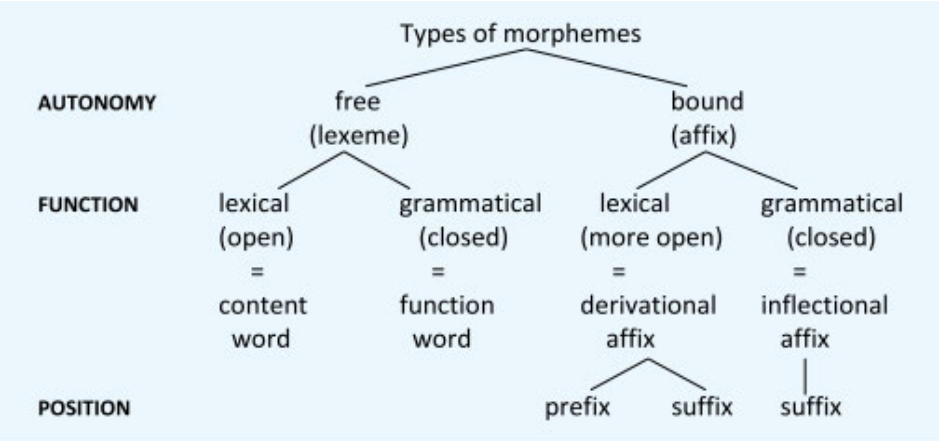

Figure 2.1: Clustering of different types of morphemes [23, p. 53]

Using lexical basics, *phrases* and *clauses* can be formed. While phrases can have a complexity as low as only containing a single word, clauses employ a subject-predicate structure. Clauses are distinguishable in *finite* and *non-finite* clauses, where the former uses a tensed predicate and the latter is only used in subordinate clauses and formed using the verb's infinite form with "to", a present participle or a past participle. [23, pp. 90-91]

The last linguistic branch considered, *semantics*, deals exclusively with the meaning of linguistics. Semantic studies address not only the deduction of a lexeme's meaning from its lexical form but also various ways of analyzing meanings and semantic structures. [23, p. 143]

The term *synonymy* describes semantic similarity or even equivalence using different lexemes. Synonyms are broken down into *descriptive* and *total* synonyms. Descriptive synonyms, although being exchangeable, might alter the meaning in regard to connotation, stylistic level or regional as well as cultural meaning. On the other hand, total synonyms allows two lexemes to be exchanged without changing the meaning in any context. [23, pp. 151-152]

Contrary to synonyms, *antonymy* means an opposite term for a given lexeme. Antonyms are split up into *complementary antonyms* forming a binary word pair representing direct negations (such as alive and dead), *gradable antonyms* where contrary lexemes can be sorted systematically based on their context, *relational opposites* referring to counterparts in a specific relation or comparison (such as teacher and pupil) and *directional opposites* which include direction-based lexemes which represent antonyms (such as left and right). [23, p. 152]

2.2.2 Linguistics in Prompt Design

Examining many of the aspects summarized in Section 2.2.1, Leidinger *et al.* conducted a broad study on how linguistic variations in a prompt influence the text outcome of LLMs within specific task scenarios. In general, it is evident that linguistic patterns influence the generation quality. While prompts formulated as questions or orders tend to lead to better results than simple statements, the usage of aspect (active or passive) as well as tense does not influence outcomes as heavily. Their linguistic studies were also able to show that exchanging modality words and phrases as well as entire words with synonyms surprisingly results in performance changes and especially

enhancements when using unusual phrases. However, their work was not able to detect major performance changes when increasing prompt or model sizes, therefore overall emphasizing the advantages of utilizing linguistic properties within prompt design instead of the model itself. [24]

Another point of interest is the grade of politeness used within prompt design. In general, politeness does not significantly influence task performance for LLMs. However, notable changes in content length could be observed when prompting in English language. While some models tend to generate shorter texts when using impolite formulations, other state-of-the-art LLMs such as GPT-3.5 or Llama2 show longer outputs when using impolite prompting. Even more complex models such as GPT-4 seem to completely ignore impolite formulations and instead focus on the task exclusively. [25]

2.2.3 Scraping of Linguistic Components

Scraping refers to "the process of extracting, copying, screening, or collecting data" [26, p. 8]. To collect a wide variety of linguistic components along with synonyms, scraping processes can be put into use to fetch and cluster linguistic terms.

This work aims to utilize *thesauruses* to automate the process of scraping sets of words. A thesaurus serves as a language reference, grouping words with similar meanings. Different variants of thesauruses can be arranged as follows: [27]

- **Roget:** Thesauruses with Roget's style provide a useful aid for writers to select meaningful words, supplying many word variations (see [28] for reference).
- **WordNet:** An open-source database with English words clustered by psycholinguistic criteria. Therefore, WordNet provides synonyms and hyponyms for nouns and verbs and antonyms for adjectives. Additionally, terms are sorted into *synonym sets* which are linked depending on the words' meanings while mostly keeping different parts of speech distinct from each other. [29]
- **IR-manual:** Domain-specific thesauruses, similar to the basic structure of WordNet utilizing synonymy and taxonomy. Often, generalization with broader, narrower or related terms are given.
- **Automatic:** Automatized approaches statistically analyze word distributions in text corpora, with the most common mathematical representation being vectors. This way, various algorithms can be applied to classify words and establish affinities between words.

2.3 Evaluating Text Quality using Readability Metrics

Evaluating the quality of texts generated by LLMs is a crucial aspect to measure the performance of a model. Following the taxonomy of Guo *et al.*, it is possible to classify model evaluation into five major categories: [30, pp. 6-8]

- **Knowledge and Capability:** Measurements for general LLM estimations as well as reasoning aptitudes and specific question-answering capacities.
- **Alignment Evaluation:** Evaluations of ethical concerns behind content generated by LLMs.
- **Safety Evaluation:** Assesses the robustness and potential towards Artificial General Intelligence (AGI) of a model.

- **Specialized Evaluation:** Evaluations for specific domains, such as finance, computer science or legislation.
- **Evaluation Organization:** Different means of quantifiable measuring model performance to compare LLMs on an objective base.

As this work aims to improve content generation in terms of text readability, the evaluation metrics settled in the area of *Evaluation Organization* will be prioritized. Therefore, some of the most common evaluation metrics when it comes to assessing text quality will be taken into consideration. For this work, this namely are various *readability metrics* relying on statistical appraisals. As the following enumeration shows, each index uses different calculations operating on the layer of words or sentences:

1. **Flesch-Kincaid Readability Score (FK)** [31]: The Flesch-Kincaid readability metric offers a numerical assessment of the readability of a text based on its average sentence and word length. The formula for computing the Flesch-Kincaid readability score is:

$$FK = 206.835 - 1.015 \left(\frac{\text{total words}}{\text{total sentences}} \right) - 84.6 \left(\frac{\text{total syllables}}{\text{total words}} \right)$$

   Initial Usage and Utility: Initially developed for usage in the U.S. Navy, it is widely used to assess the readability of educational materials and to ensure that texts match the reading abilities of the intended audience.
2. **Coleman-Liau Readability Index (CLI)** [32]: The Coleman-Liau readability index determines text readability based purely on the number of characters, words, and sentences. This index is calculated using the following formula:

$$CLI = 0.0588 \left(\frac{\text{total letters}}{\text{total words}} \times 100 \right) - 0.296 \left(\frac{\text{total sentences}}{\text{total words}} \times 100 \right) - 15.8$$

   Initial Usage and Utility: CLI is particularly useful for automated readability assessments due to its reliance on easily countable characters rather than syllables.
3. **Dale-Chall Readability Formula (DC)** [33]: The Dale-Chall readability formula incorporates a list of familiar words to assess text readability. The corresponding score is calculated as follows, where “difficult words” refer to terms not included in the pre-defined list of familiar words:

$$DC = 0.1579 \left(\frac{\text{difficult words}}{\text{total words}} \times 100 \right) + 0.0496 \left(\frac{\text{total words}}{\text{total sentences}} \right)$$

   Initial Usage and Utility: This metric emphasizes the importance of familiar vocabulary in determining readability, making it particularly useful for educational texts aimed at younger readers.
4. **Gunning-Fog Index (GF)** [34]: The Gunning-Fog index evaluates text readability by considering the average sentence length and the percentage of complex words, where terms with three or more syllables count as “complex”. The Gunning-Fog index is calculated using the following equation:

$$GF = 0.4 \left[\left(\frac{\text{total words}}{\text{total sentences}} \right) + 100 \left(\frac{\text{complex words}}{\text{total words}} \right) \right]$$

   Initial Usage and Utility: Created by Robert Gunning in 1952, it aims to make texts more accessible by identifying and reducing complex words and long sentences.

5. **Simple Measure of Gobbledygook ($SMOG$)** [35]: The SMOG index approximates the number of years of education needed to understand a text based on the amount of polysyllabic words. The formula is:

$$SMOG = 1.0430\sqrt{\text{number of polysyllabic words} \times \left(\frac{30}{\text{number of sentences}}\right)} + 3.1291$$

   Initial Usage and Utility: SMOG is widely used in health communication to ensure that health information is accessible to a broad audience.
6. **Automated Readability Index (ARI** [36]: The ARI uses characters per word and words per sentence to estimate the readability of a text. The formula is:

$$ARI = 4.71\left(\frac{\text{total characters}}{\text{total words}}\right) + 0.5\left(\frac{\text{total words}}{\text{total sentences}}\right) - 21.43$$

   Initial Usage and Utility: Although it was initially developed for military manuals, *ARI* is particularly useful for automated readability assessment due to its reliance on easily countable characters.
7. **Läsbarhetsindex (LIX** [37]: The Lix index is a readability measure based on the average sentence length and the percentage of long words (more than six letters). The formula is:

$$LIX = \left(\frac{\text{total words}}{\text{total sentences}}\right) + 100\left(\frac{\text{long words}}{\text{total words}}\right)$$

   Initial Usage and Utility: Developed in Sweden, Lix is used primarily in Scandinavian countries to assess the readability in various languages. It is effective in differentiating texts with varying levels of complexity.

2.4 Reinforcement Learning

As defined by Sutton and Barto, RL is “learning what to do — how to map situations to actions — so as to maximize a numerical reward signal”. [16, p. 1]

The general idea behind RL is to make a learning subject, the so-called *agent*, automatically select fitting *actions* in a dynamic *environment*. Based on their behavior, the learner will receive *reward signals* which provide information on the valence a certain selected action had. Using these rewards, the learner adapts to the environment and becomes more likely to choose actions that lead to higher rewards. Because of the described details, RL, although being a subfield of ML, is fundamentally different from other procedures such as *supervised learning* or *unsupervised learning*. [16, pp. 1-3]

An important dilemma in RL deals with the *tradeoff between exploration and exploitation*. This problem is a matter of enabling the agent to use available actions to find novel ways of dealing with the given problem (therefore, to *explore* the environment) while at the same time using promising actions explored in the past to maximize outcomes (in other words, *exploiting* the environment). There are various strategies appliable to support both learning about the environment and exploiting favor actions. [38, p. 16]

2.4.1 Markov Decision Process

A MDP is an abstract framework used to model stochastic decision-making and is therefore suitable for many RL algorithms. Following the definition of Hu, MDPs are commonly modeled by the following components: [38, pp. 21-22]

- **States** ($\mathcal{S}$): The set of all possible configurations that the environment may take on.
- **Actions** ($\mathcal{A}$): The set of all actions the agent can select to interact with the environment.
- **Reward function** ($\mathcal{R}$): The function providing the reward signal for selecting a certain action a in state s where $s \in \mathcal{S}$ and $a \in \mathcal{A}$
- **Transition model** ($\mathcal{P}$): The function modeling the probability of the environment transforming to state s' after taking action a where $s' \in \mathcal{S}$ and $a \in \mathcal{A}$. The model $\mathcal{P}$ is often not considered explicitly, as the environment depicts this function implicitly due to its reactions.

In the context of RL, a MDP can be used to represent the training loop in which an agent learns to select an Action $a \in \mathcal{A}$ depending on the current state $s \in \mathcal{S}$. Based on the action, the environment responds with a reward that the agent can use to adjust its future decision-making in order to make high rewards more likely. In terms of the process, each discrete time step can be indexed with a parameter t where $t \in \mathbb{N}$ with the initial state in which $t = 0$. Formally, each taken action A_t in state S_t leads to a new state S_{t+1} and a reward R_t, as schematically shown in Figure 2.2. [38, pp. 21-23]

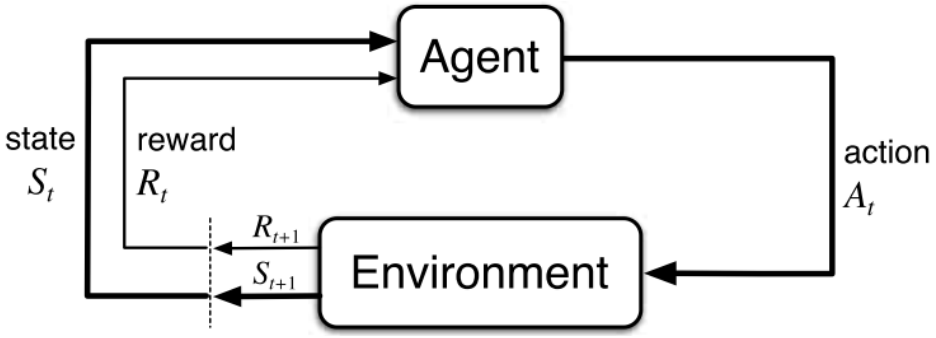

Figure 2.2: Schematic interaction loop of an agent with the environment in a MDP [16, p. 48]

In the following, other important terms for MDPs will be covered. MDPs can be characterized by the so-called *Markov Property* which ensures that "the future state of a system is independent of the past given the present" [38, p. 25]. In other words, in a MDP the next state will be determined only by the action selected in the current state.

The *return* G_t of a RL iteration episode describes the sum of rewards from time step t to the end of the episode T: [16, p. 54]

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.1}$$

To increase reward values occurring closer to the time step t, the return is often extended by the concept of a *discount rate* γ:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + R_T \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{2.2}$$

where the parameter γ $(0 \leq \gamma \leq 1)$ is called the *discount rate*. [16, p. 55]

The function which determines the next action to be selected is called a *policy* π, where $\pi(s)$ determines the probability for each action an agent can take while being in state s. To optimize a policy, the agent estimates a *value function* V to predict how good a given state is by considering the expected rewards for future steps when following a given policy π. [16, p. 58]

The value function is computable in a recursive formula called the *Bellman expectation equation* which takes into account the immediate reward, as well as all the expected successor rewards discounted by the discount rate γ: [38, p. 31]

$$\begin{aligned} V(s) &= \mathbb{E}\Big[G_t | S_t = s\Big] \\ &= R(s) + \gamma \sum_{s' \in S}^{\infty} P(s'|s) V(s') \text{ , for all } s \in \mathcal{S} \end{aligned} \tag{2.3}$$

Relying on this concept, the value function for a given state s under policy π is computed using:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\Big[G_t | S_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_t + \gamma V_\pi(S_{t+1}) | S_t = s\Big] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \Big[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s')\Big] \text{ , for all } s \in \mathcal{S} \end{aligned} \tag{2.4}$$

Regarding a RL problem, the agent tries to find the optimal policy π_*. [16, p. 62]

2.4.2 Temporal Difference Learning

Temporal Difference Learning (TDL) is a RL learning technique that adjusts the agent's behavior by comparing the predicted and actual rewards received after taking a certain action in a given state. TDL is considered a *model-free* method, as the agent does not need an explicit model of the environment to update its policy. [38, pp. 75-76]

In TDL, the value function is optimized by updating estimates considering the *temporal difference error* between the calculated and actual rewards from a state S_t selected under policy π using the following formula:

$$\begin{aligned} V_\pi(S_t) &= V_\pi(S_t) + \alpha\Big(G_t - V_\pi(S_t)\Big) \\ &= V_\pi(S_t) + \alpha\Bigg(\Big[R_t + \gamma V_\pi(S_{t+1})\Big] - V_\pi(S_t)\Bigg) \end{aligned} \tag{2.5}$$

where α is considered the *learning rate*. The learning rate determines the magnitude of parameter updates and critically influences the convergence speed and stability of the training process. A lower learning rate ensures smoother optimization trajectories but may prolong training time, while a higher learning rate risks overshooting the optimal solution. [38, p. 77]

2.4.3 Q-Learning and Deep Q-Learning

Q-Learning is a special implementation of TDL which optimizes towards an optimal *state-value function* called Q_π for a given policy π. Q takes in a pair of state s as well as the taken action a and estimates the pair's quality based on the expected succeeding return of the episode: [38, pp. 36-37]:

$$\begin{aligned} Q_\pi &= \mathbb{E}_\pi\Big[G_t | S_t = s, A_t = a\Big] \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a') \text{ , for all } s \in \mathcal{S}, a \in \mathcal{A} \end{aligned} \tag{2.6}$$

Q-Learning tries to estimate the optimal state-action value function Q_* by incrementally updating the state-value function using the learning rule of TDL as suggested by Watkins and Dayan: [39]

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(\Big[R_t + \gamma \max_{a'} Q(S_{t+1}, a')\Big] - Q(S_t, A_t)\right) \tag{2.7}$$

Estimating a fitting Q-function is especially difficult when dealing with complex environments or large action spaces. Therefore, Deep Q Networks (DQNs) are often used to leverage *deep learning* to estimate an optimal Q-function. According to Ketkar and Moolayil, deep learning is a "subfield within machine learning that deals with the algorithms that closely resemble an over-simplified version of the human brain that solves a vast category of modern-day machine intelligence" [40, p. 1].

Deep learning therefore focuses on abstracting data representations to conclude features so that knowledge for new data can be acquired automatically. To fulfill this task, deep learning often utilizes *neural networks* which provide a concept representing the human brain mathematically. Therefore, so-called *neurons* are organized in different layers, with exactly one input- and one output layer and any number of so-called hidden layers in-between. Neurons between the layers are connected with *weights* which will be updated during training to reduce error and make the output layer provide the most accurate results possible. [40, pp. 7-8, pp. 93-96]

In the context of RL, DQNs are used to express the state-action value function Q_π by passing the current state to the input layer and using the output layer to predict the Q-values for all possible actions the agent can select from. The loss used to update weights in the network is calculated from the error between the predicted values and actual targets of TDL. [38, pp. 146-147]

2.5 Combining Reinforcement Learning and Prompt Engineering

As of the current state-of-the-art, little effort has been made to combine RL on the high level of generating prompts for LLMs.

For now, RL is mostly incorporated when training LLMs. As an example, *InstructGPT* offers a fine-tuned model closely aligned to other GPT models using human feedback. In their approach, Reinforcement Learning from Human Feedback (RLHF) is used to guide output towards human-preferred samples. Research showed that content generation by a model fine-tuned using this method leads to more favorable results, although using significantly fewer parameters. [41]

The research of Deng *et al.* is particularly interesting for this work, as the authors were able to demonstrate that RL is usable to generate prompts fine-tuned for certain problems of NLP. Their implementation called *RLPrompt* uses frozen LLMs to generate tokens then filled into a pre-defined mask used as prompt input. Results imply that not only task-specific results greatly improve, but generated prompts are also transferable to other LLMs. This observation is substantiated by the fact that some tokens favored by the RL agent consist of contextless gibberish text and symbols, proposing that LLMs use an individual grammatical structure. [42]

2.6 Distributed Systems

Distributed systems represent a concept that has gained increasing importance in the context of complex applications. According to Tanenbaum and van Steen, due to the variety of different manifestations, the topic can be broadly defined as a collection of independent computers that appears to its users as a single coherent system. [43, p. 19]

This section focuses on presenting goals of distributed systems and classifying them in comparison to stand-alone services. In addition, *RESTful services* are presented as a paradigm for the realization of distributed systems within this work.

2.6.1 Fundamental Goals and Basic Types of Service Architectures

In the following, the core aspects that make up a distributed system will be examined and highlighted through their fundamental concepts and objectives. As noted in the definition by Tanenbaum and van Steen, a distributed system can only exist through the cooperation of multiple autonomous components that collectively represent its functionality. Coupling individual elements across existing boundaries enables cooperation or integration within the overall system. [44, p. 5]

The most important goal is to simplify access to remote resources, allowing for mutual control and efficient shared use. Besides the obvious savings from shared usage, this connection also facilitates collaboration and information exchange. With extensive networking, as seen with the Internet, numerous security requirements arise, which are also relevant in distributed systems. Concerns such as the secure transmission and storage of sensitive data often highlight gaps in user authentication or unwanted communication. By incorporating services based on predefined criteria such as syntax and semantics, the goal of an open distributed system can be achieved. Specifying interfaces allows different processes to communicate and be standardized within the system. These interfaces can be implemented in various ways, giving different implementations equal status within the distributed system. Openness also includes easy extensibility through interchangeable components that can be integrated into the system by different developers. Such components should be addable or removable without affecting existing functions. [43, pp. 20-23]

Due to its properties, a distributed system can be seen as a counterpart to *monolith* systems, which are defined as an environment where all components run in a single process communicating internally. Both distributed systems and monoliths can be referred to as a Big Ball of Mud (BBoM) when exhibiting bad conceptual architecture. [45, p. 123, p. 179]

Monolithic architectures can still occur in a modular structure, often being named as *moduliths*. Systems like this pair simplified internal communication processes with better possibilities to prevent software erosion. Other monolith hybrid optimizations focus on embedding components, which are usually found when developing solely distributed systems. One example is found when combining *microservices* with a monolithic core. Microservices are another type of architecture style, leveraging multiple independent software systems with individual, independent areas of responsibility each to fulfill more complex tasks when networking. To allow seamless communication processes, interaction should not depend on specific programming languages or frameworks but on exactly specified communication interfaces instead. By combining microservices with a monolithic core, the foundation of the architecture can maintain a great scale of consistency, allowing other parts of the system to leverage the advantages of distributed systems using microservices. [45, pp. 169-171, pp. 179-181]

The two previously outlined approaches are visualized in 2.3 based on Dowalil's distinction.

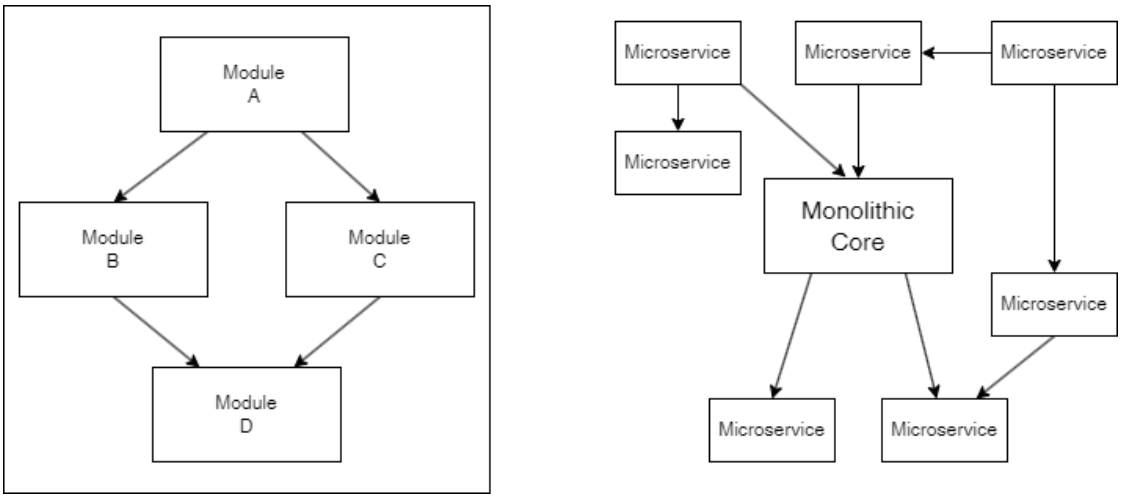

Figure 2.3: Frameworks for a modulith (left) and a hybrid microservice approach with a monolithic core (right) (own representation following [45, p. 179, p. 181])

2.6.2 RESTful Services

Representational State Transfer (REST) is an architectural style for distributed systems, particularly suitable for the web. Unlike other networked software architectures that focus on standardizing messages and protocols, REST emphasizes the logic behind modern web architecture, proposing a set of design principles that naturally align with the web. At its core, REST is not a protocol or a specification, but a set of guidelines for exposing and manipulating resources over a network using the Hypertext Transfer Protocol (HTTP). Core aspects of REST include resource identification, where each resource is uniquely identified by a Uniform Resource Identifier (URI), and the use of a unified resource interface, enabling client applications to interact with resources via standard HTTP operations (such as GET, POST, PUT, DELETE) to access or mutate data using the interface. Additionally, REST supports the concept of links and hypermedia, allowing resources to be interconnected, thus

facilitating client navigation between different resource states. A resource in REST is defined as a uniquely identifiable model with one or more representations, which can be transmitted in various data formats. [46, pp. 67-68]

For object representation, JavaScript Object Notation (JSON) has emerged to be a lightweight, easy-to-read and -write data format in the recent years. With significantly less verbose overhead than previously used formats like XML, not only data size is reduced, but interface responses are far less difficult to parse. With many practical libraries in common programming languages, JSON can be easily read, which is handy when it comes to response interpretation and consequent reuse in the implementation context. [47, pp. 36-40]

3 Methodology

At its core, the methodology aims to dynamically generate fillings for spaces in pre-defined prompt templates to improve the metric-assessed readability of LLM-generated texts. To allow the greatest grade of adjustment possible in terms of the complex action space the problem provides, the methodology aims to incorporate a flexible representation for both the used prompt templates and the possible terms filled in for the final prompt. For this purpose, RL will be leveraged to explore dynamic strategies in order to fill these templates for maximum readability outcome.

RL is particularly well-suited for this objectives due to its ability to adaptively optimize decision-making processes in complex environments, which directly aligns with the dynamic nature of prompt engineering for LLMs. In the context of this research, reward signals are directly tied to the readability quality of the generated text, measured by readability metrics. The integration of RL into prompt generation is advantageous because it allows for continuous learning and improvement based on feedback. Unlike static rule-based systems, RL agents can explore a vast space of potential prompt configurations, discovering combinations that may not be immediately intuitive but yield superior results. This exploration-exploitation tradeoff enables the system to balance the need for trying new strategies with leveraging known successful patterns. This dynamic adaptability is critical in the ever-evolving landscape of language models, where optimal prompt structures can vary widely depending on the specific context and desired output. Furthermore, RL's capacity to operate within a MDP framework allows the system to model the prompt generation process as a series of decisions, where each step is informed by the current state of the prompt. This sequential decision-making process mirrors the way human designers iteratively refine a prompt, but with the added benefit of automation and scalability. By leveraging techniques such as Q-learning or Deep Q-learning, the system can effectively handle the large action spaces involved in prompt engineering, selecting the most appropriate words and phrases to fill in predefined templates, thereby optimizing the prompt's effectiveness.

The applied methodology is heavily based on the research of Deng *et al.*, who have already showed that prompt engineering is optimizable for specific tasks using RL (see Section 2.5 for reference). However, this research aims to differentiate in two key aspects:

1. **Discrete action space**: Instead of using frozen LLMs to generate continuous tokens for prompt generation, the methodology of this paper relies on a *word pool*. A RL algorithm is then used to optimize the discrete selection to assemble prompts that have a high potential to generate appropriate content.
2. **Focus on text readability and content coherence**: Rather than optimizing prompts for task specific scenarios, the main goal is to show that a semi-automated process utilizing RL is capable of improving the quality of generated texts. The metrics that are employed will focus on text quality and general coherence to address generic requirements users have when using LLMs to generate text content.

In this chapter, after the central questions and hypothesis of this work are addressed, the general methodology is outlined. Using a simplified approach to the workflow, a preliminary experiment is conducted to test the limits of the general idea. Based on the main findings, an improved methodology is presented that includes several refinements of the initial approach. Based on important identified variables, further hypotheses are formulated.

3.1 Central Question and Hypothesis

The central question of this thesis revolves around the application of reinforcement learning RL to improve the readability of text generated by LLMs. The primary scientific question investigated is whether a RL-based prompt engineering approach can effectively enhance text readability compared to traditional, non-optimized methods. This study explores the use of a discrete action space for RL, where an agent selects words from a predefined pool to fill specific template slots, aiming to optimize the output's readability.

To address this question, the thesis builds upon existing research in RL and prompt engineering, specifically focusing on optimizing prompts for text quality rather than task-specific scenarios. By leveraging the dynamic adaptability of RL and employing metrics that assess text quality and coherence, this work seeks to demonstrate the potential of RL in refining prompt engineering for better readability outcomes.

For this question, an initial hypothesis H_1 is formulated.
H_1: RL improves text readability when comparing prompts using composition guided by a trained RL agent and randomized combinations.

Section 3.3 will provide a prototypical methodology implementation used to evaluate the initial hypothesis.

3.2 Outline of the Approach

The workflow can be summarized in an optimization loop which maximizes text readability by assembling a textual prompt used for inference with a high-level LLM interface. The input to the procedure could be a single term or a wording phrase that describes the topic which the generated text will be about.

A predefined *prompt template* is then used to assemble the final prompt. The template consists of a static textual base which contains various *mask spaces* at likewise manually set positions. Mask spaces can be filled out by automatic procedures to determine the content of the full prompt.

For this work, RL algorithms will be used to select words that will be inserted into the mask spaces. RL is used in this work to dynamically optimize prompt generation for several reasons. Firstly, RL allows adaptive learning, where an agent can continuously improve its decision-making process based on feedback from previous actions, which aligns with the goal of enhancing text readability. Secondly, RL's ability to handle large and complex action spaces, especially in combination with deep neural networks, makes it well-suited for selecting words from a predefined pool to fill specific

template slots, thereby optimizing the quality of generated text [48]. Lastly, RL enables exploration of various combinations of words and phrases, allowing the system to discover prompt configurations that lead to better readability results, which would be challenging to achieve using static, rule-based approaches. The concrete RL implementation as well as the process of inserting selected terms in the mask spaces varies between the different approaches, which will be introduced in this chapter.

After applying the filling method, the full prompt will then be sent to a LLM that is used to generate text output. To deduce a numeric reward that can be used to train the RL agent, the text is scored using a separate evaluation step. This work focuses on *readability metrics* to assess the quality of generated text and therefore the reward representation. The reward signal ultimately is used to perform one training step on the RL agent. In other words, each text sample generated from the agent's mask selection will then be scored to make the agent more likely to select single mask entries or entire combinations that lead to higher text quality based on the metrics used for evaluation.

The whole process from input of one term to training the RL algorithm to maximize text quality in terms of readability using a word pool is visualized in Figure 3.1.

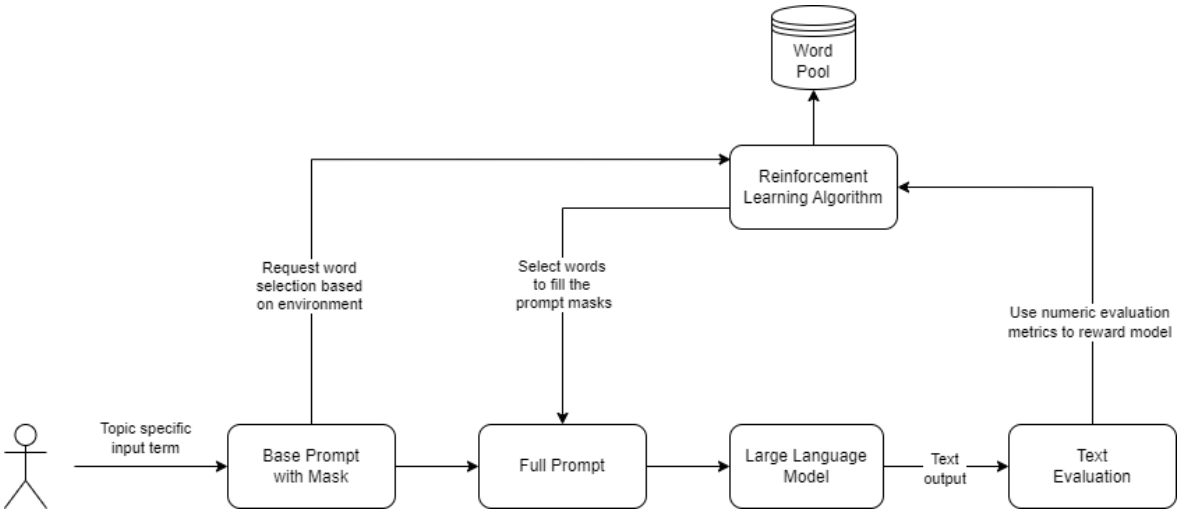

Figure 3.1: Process visualization for the used methodology

3.3 Preliminary Experiment

To initially assess whether RL is capable of increasing text quality in the scenario of semi-automated prompt engineering as given in the workflow description in Section 3.2, a preliminary experiment has been conducted. The main goal is to apply a highly simplified approach of the methodology to accept or reject hypothesis H_1.

For the input term used for text generation, a static list will be used. While in a production environment this input could be user-defined, for the given experiment purposes there will be a defined list of topics which will be used to simulate user choice.

3.3.1 Reinforcement Learning Model Engineering

The RL process to fill in the mask with terms from the word pool will be abstracted to a MDP modelled with the following properties:

- **State space** $\mathcal{S}$: In the context of the problem, each state represents the current stage of the prompt, including the partially filled prompt mask in terms of its mask spaces.
- **State space** $\mathcal{A}$: The set of actions the agent can select from, where each action a will be mapped to a tuple (w, b) under word pool $\mathcal{W}$ (see Section 3.3.2 for details) where b is a boolean flag that determines whether another word will be filled into the mask after the current selection or not. $(w \in \mathcal{W}, b \in \{0, 1\})$
- **Reward Function** $\mathcal{R}$: Each generated text sample is evaluated using numeric metrics. Using normalization in the context of each chosen scoring system, it is important to ensure that reward signals will have appropriate levels. Details of this process will be discussed in Section 3.3.3.

While the space of all available actions is determined using $|\mathcal{W}| * |\{0, 1\}| = 2|\mathcal{W}|$, the iterative logic might lead to term chains of extensive length. To limit time and space complexity, each mask within the prompt template has a maximum amount of words that will be filled in.
The process ensures the Markov Property, as each state contains all necessary information to process another action selection. Using this procedure, the agent is capable of iteratively picking several terms from the word pool that make up the full prompt without needing any information from past training iterations.

For implementation, a DQN will be used to allow optimal training convergence in the large action space provided. To find a fair tradeoff between exploration and exploitation, the RL algorithm will use an *epsilon-greedy* strategy for its policy. In this technique, the policy selects the currently most favorable with probability ϵ or a random action $a \in \mathcal{A}$ with probability $1 - \epsilon$: [38, p. 71]

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{if } a = \underset{a}{\operatorname{argmax}} Q_\pi(s, a) \\ \frac{\epsilon}{|A(s)|} & \text{if } a \neq \underset{a}{\operatorname{argmax}} Q_\pi(s, a) \end{cases} \tag{3.1}$$

where a high value of ϵ $(0 \leq \epsilon \leq 1)$ will lead to exploring the environment using randomized actions whereas a small value will exploit the learned favor choices. A fair trade-off between exploration and exploitation is furthermore encouraged by *decaying* ϵ over the training episodes. This means that a high value is assigned to ϵ when starting the training and linearly decreased in every training step until a minimum end value is reached. Using this common technique, the policy will tend to explore the environment in the start whereas after training for a longer time, preferred actions are more likely to be selected.

3.3.2 Constructing the Word Pool and Prompt

The construction of the word pool plays a decisive role in the success of the RL-based text generation system. A diverse and extensive word pool enables the agent to select from a wide range of terms and phrases, easing the generation of contextually relevant text outputs, contributing to improved readability. The word pool for the experiment was constructed manually, meticulously selecting fitting terms as described in the following.

To ensure the word pool consists of a wide range of linguistic elements, various types of terms were included, ranging from nouns, verbs, and adjectives to articles, conjunctions, and prepositions.
In particular, the inclusion of simple words such as articles (e.g., "the", "a", "an") and conjunctions (e.g., "and", "but", "or") is essential for achieving smooth transitions between phrases and clauses within the generated text when filling masks with sentences or longer word groups. These basic linguistic elements serve as the anchor that binds individual words and phrases together, enabling the creation of coherent and cohesive sentences.

In addition to including different types of terms, special emphasis was put on maximizing variety and potential for phrase construction within the word pool. This involved selecting a diverse range of terms that can be combined to form semi-coherent phrases, thereby enabling the generation of contextless phrases filling the mask spaces.
For example, the word pool includes a combination of nouns (e.g. “text”), verbs (e.g. “write”, “summarize”), and especially adjectives related to text modifications (e.g. “short”, “coherent”, “readable”). These elements can then be combined by the agent to form simple constructions of no particular order and without considering linguistic patterns.

For the experiment, the initial prompt template was laid out by empirically testing out various prompt and mask combinations, generally following the *CLEAR* path. Other linguistic conditions have not been considered, as the experiment focuses only on validating the general approach of substituting given mask spaces with words from the discrete set.

3.3.3 Text Evaluation Metrics

For each generated sample, an *evaluation unit* will be used to score text quality. As a preliminary setup, the readability metrics *Flesch-Kincaid*, *Coleman-Liau*, *Dale-Chall* and *Gunning-Fog* summarized in Section 2.3 are utilized to represent an objective measure to evaluate each text sample generated by the LLM. These four metrics were selected to represent a limited subset of all considered metrics, which were examined in order to evaluate and differentiate possible improvements in the ratings in a more targeted manner.

However, to ensure an efficient learning progress, scores provided by readability metrics must be *normalized* in order to make them comparable. While normalization usually requires thorough manual tuning, for the experiment the process will be dependent on the chosen normalization implementation for readability metrics, as stated in Section 3.3.4.

To make sure the agent learns to avoid certain actions, a predefined offset value will be subtracted from the calculated mean value. This way, lower scores might result in negative rewards, leading the agent to better differentiate between good actions (positive reward signals) and undesirable selections which will be avoided in the future. The mean score for an evaluation batch P_t for text t is then calculated using the value of all normalized indices with the offset value o:

$$P_t = \frac{FC(t) + CL(t) + DC(t) + GF(t)}{4} - o \tag{3.2}$$

The computed result is directly passed as the reward to the RL agent.

3.3.4 Implementation

Python was chosen as the programming language for its widespread adoption in the field of ML and NLP. Its simplicity, readability, and extensive library support make it an ideal choice for implementing the algorithms and models described at the beginning of this section.

The word pool used for training is included in Appendix A. Prompt templates were tested out iteratively. Although results did not show significant changes judging from the few first training episodes, the final prompt template used for training and evaluation has been fine-tuned. All template iterations are summarized in Appendix B.

Language Model Implementation

For this experiment, the LLM *Vicuna* was used. Vicuna is an open-source chatbot LLM which was built from fine-tuning *LLaMA*. Therefore, about 70,000 conversations shared by other users were used for training. [49]

Different model sizes of Vicuna, including the largest variant *Vicuna-33b*, were taken into consideration in *Chatbot Arena*. This site presents users anonymized outputs of two models and lets them subjectively evaluate which text is better. Using this workflow, an elo leaderboard directly comparing different models was deduced. Moreover, text samples generated with Vicuna were assessed using OpenAI's GPT-4 model, underlining the point that samples generated by Vicuna provide solid outcomes in terms of important determinants such as relevance and detail. Using any of the presented evaluations, Vicuna shows considerable results, especially when being compared with commercial models such as OpenAI's GPT-models or Claude. [50]

Vicuna was used due to its trade-off between solid performance and at the same time resource-efficiency. This allowed to set up a smaller model variant, namely *Vicuna-7b v. 1.5*, in a local environment. Using an open-source model also caused no further costs when using inference operations. This was of great importance as for RL training, thousands of samples are necessitated. Moreover, Vicuna supports access using the OpenAI API standard, allowing to switch to another LLM at a later point in time when needed for additional experiments.

Used Frameworks

The implementation leveraged several open-source packages to facilitate various aspects of the RL learning-based text generation system:

- **PyTorch**[3]: *PyTorch*, a popular deep learning framework, served as the backbone for implementing the DQN model architecture. PyTorch's dynamic computation graph and GPU acceleration capabilities were essential for efficiently training the neural network model for text generation. PyTorch's flexibility and ease of use enabled rapid prototyping and experimentation, which was important for iteratively improving the RL algorithm.

[3] https://pytorch.org/

- **Requests**[4]: The *Requests* library provided a convenient interface for sending HTTP requests to interact with *Vicuna* for text generation. Using the OpenAI API standard, requests were sent using a JSON request body providing relevant information such as the target model and the input prompt.
- **Readability**[5]: The Readability library played a pivotal role in evaluating the generated text's readability and coherence using the named readability metrics (see Section 2.3 for reference of the selected metrics). By using the package's functionality, the system assessed the linguistic quality and readability of the generated text samples, providing valuable insights into their suitability for human consumption and therefore the reward function used to train the RL agent. The package is also mandatory for normalizing scores to make the different metrics used directly comparable.

Initially, the implementation of the RL-based text generation system began with an intuitive and straightforward DQN approach. The model served as the start-point for training the text generation model, with a focus on learning a policy to optimize the selection of words for prompt completion.

The simple DQN architecture consisted of a neural network with fully connected layers, mapping the state space of the environment to action values corresponding to word selections (see Section 3.3.4 for details). The agent interacted with the environment by selecting actions (i.e., choosing words to fill mask spaces) and receiving rewards based on the quality and coherence of the generated text. The training process involved iteratively updating the DQN parameters using gradient descent to minimize the discrepancy between predicted and target Q-values.

While the initial DQN approach showed some promise in improving text generation quality, it exhibited some instability and variability in training results. The learning process often suffered from issues such as overestimation bias in Q-value estimates, leading to suboptimal performance and slower convergence (details are shared in Section 3.3.5). These issues will be addressed in Section 3.4.

To further simplify appraising the generated samples, the implementation includes a serialization logic which stores each training episode in a JSON object. Serialization includes the hyperparameter configuration, especially the current value of ϵ, as well as the LLM prompt input and corresponding output. After passing the output through the evaluation unit, each separate score as well as the overall reward resulting from the scores' mean value were saved as well. A sample for one training episode serialized to JSON is given in listing 3.1. As masks in the prompt template were filled with words from the word pool, the input prompt might look uncommon. As a sample input, the given term for the training progress shown was about "Cats".

[4] https://pypi.org/project/requests/
[5] https://pypi.org/project/readability/

Listing 3.1: Sample for an episode serialized in JSON format during training (input and output texts were shortened)


```
{
  "episode": 1520,
  "epsilon": 0.2077575751215263,
  "input": "from short up. Generate a coherent away text about Cats.[...]",
  "output": "Cats are domesticated mammals that are popular pets around [...]",
  "score_normalized_mean": 1.002338991416309,
  "score_mapping": {
    "Flesch-Kincaid": 11.394472103004293,
    "Coleman-Liau": 10.739742489270384,
    "Dale-Chall": 9.0615791416309,
    "Gunning-Fog": 12.813562231759612
  }
}
```


Summary of Hyperparameters

The hyperparameters chosen for training the RL model played a crucial role in determining the training dynamics and convergence behavior. Hyperparameters were carefully tuned through iterative experimentation to achieve a balance between computational efficiency and training effectiveness. The summary of the chosen hyperparameters is as follows:

- **Batch Size**: The batch size was set to 128 to balance computational efficiency and stable training. A larger batch size facilitates faster convergence, but may increase memory requirements and computational overhead. Conversely, a smaller batch size may lead to more stochastic updates and slower convergence.
- **Learning Rate** α: A learning rate of 0.001 was chosen for efficient convergence of the neural network weights.
- **Reward offset**: An offset of 10.5 was empirically determined for the training process. Using this value to subtract from the scores' mean value leads to both positive and negative reward signals.

The neural network used as the DQN for training consists holds two fully connected hidden layers with ReLU activation functions and 128 neurons each. The input layer size is determined by the environment, namely the amount of masks given in the prompt template. The output layer size depends on the amount of terms in the word pool and therefore represents the Q-value distribution.

3.3.5 Results

This section will summarize the most important data findings and general results that were investigated in the experimental approach.

Training Results

The first training was conducted using the intuitive Q-Learning approach described earlier. Figure 3.2 shows the reward distribution as obtained from the mean of all used readability metrics over the curse of 10,000 training episodes. The orange line shows a stabilized reward trend by smoothing 20 reward samples to their mean value.

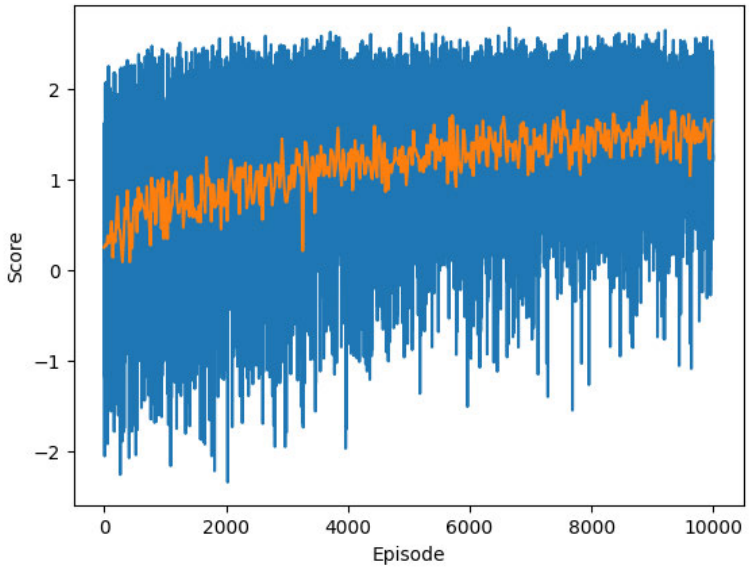

Figure 3.2: Score distribution over a training run of 10,000 episodes: All data points (blue) and the baseline composed of 20 consecutive score means (orange)

Figure 3.3 shows the distribution of the different readability metrics used to determine the overall reward for each training episode.

Evaluation: Comparing Random Prompts with Agent Selections

In order to verify that the implementation does not only achieve higher-level results in training but also in practical comparison with approaches without RL, an inference was carried out with different text generation modalities and then compared with each another. This is done in order to accept or reject hypothesis H_1. The three modes of operation that were scrutinized were:

- $\epsilon = 1$: The RL agent was used to select completely random actions, therefore filling masks with randomized terms from the specified word pool.
- No prompt mask: The prompt masks were left out of the sample generation, so the LLM only obtained the raw prompt template without any additional terms from the word pool.
- $\epsilon = 0$: The RL agent trained with the DQN under a target network was used to exploit the learned policy fully. Therefore, for each sample an action was chosen using a weighted softmax action selection, fully depending on the implementation of the experimental approach of this section. The samples generated from this method represent the best possible prompt inputs.

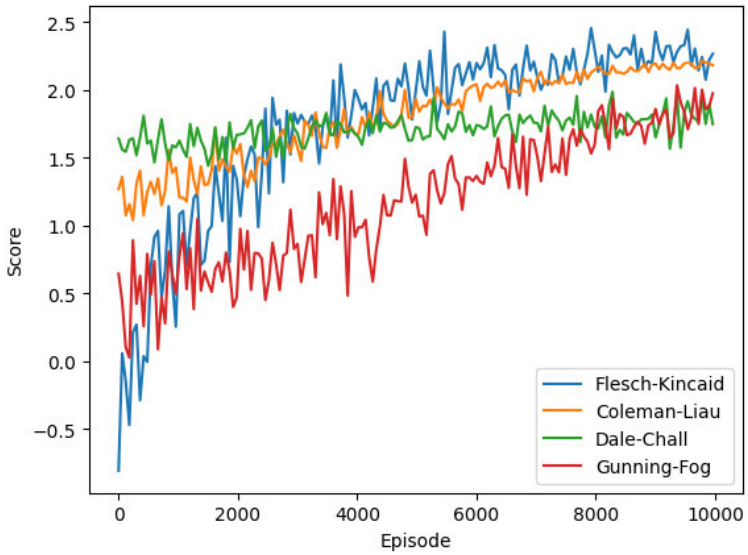

Figure 3.3: Comparison of readability measure scores used during model training over 10,000 episodes, smoothed for each 20 consecutive data samples

Using these modalities, 300 text samples were generated and evaluated with the same metrics as discussed in Section 2.3 for each of the three approaches. Direct comparison between the maximum exploit of the agent (orange) and the two approaches not taking use of RL (blue) are visualized in Figure 3.4. Statistical data for all three conducted inference runs is shown in table 3.1.

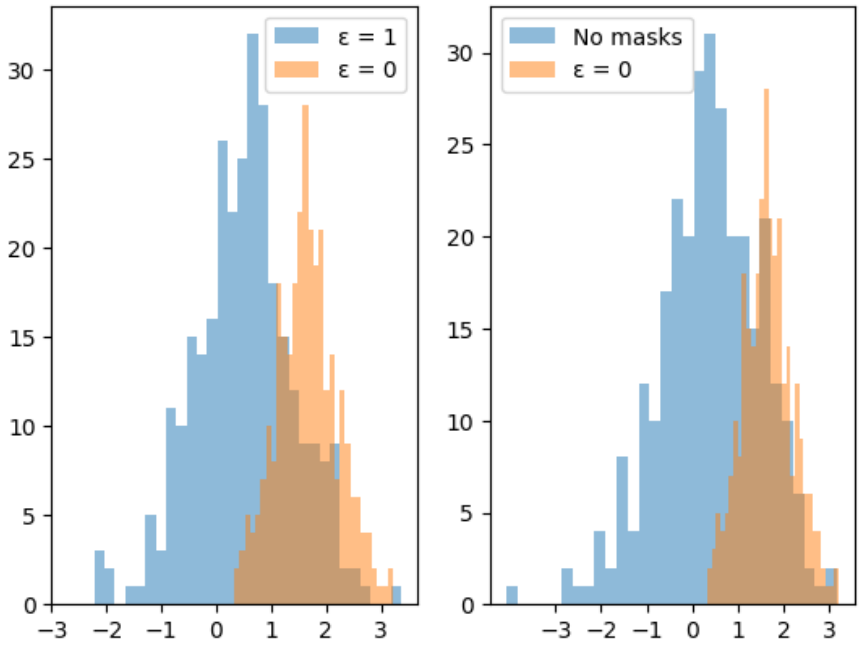

Figure 3.4: Score distribution between the implemented reinforcement learning approach (orange) and prompts not using the optimized term selection (blue)

A t-test with a significance level of 0.05 comparing the results of both test groups is conducted to verify or falsify the hypothesis H_1. Calculations result in a value of $5.891 * 10^{-}62$ for p, resulting in a rejection of the null hypothesis. Thus, H_1 is accepted, indicating significant impact of the used RL algorithm on text readability outcomes.

Table 3.1: Comparison of statistical data between prompt generation with completely random insertions, no masks and the implemented reinforcement learning approach (optimal values in bold font)

	Mean	25%	Median	75%	St. Deviation	Min	Max
$\epsilon = 1$	0.51	-0.03	0.53	1.03	0.91	-2.22	**3.38**
No prompt mask	0.37	-0.28	0.41	1.09	1.11	-4.07	3.17
$\epsilon = 0.1$	**1.66**	**1.25**	**1.63**	**1.99**	**0.55**	**0.33**	3.21

3.3.6 Discussion

The results of the experiments shed light on the effectiveness of incorporating RL techniques in improving the quality and stability of text generation models. This discussion delves into the implications of the training results and the broader implications for NLP tasks.

Impact of Reinforcement Learning on Text Generation

The training results, along with the acceptance of H_1 due to the separate evaluation samples, indicate a notable improvement in the quality stability of generated text when using RL compared to conventional methods. By employing a DQN architecture, the RL-based text generation system learns to optimize the selection of words for prompt completion, resulting in more coherent and contextually relevant outputs according to the used readability metrics. Since this turns to be true for the simplified preliminary approach, an even higher grade of optimization is expected for the full methodology implementation.

One significant observation is that the RL approach tends to produce better texts more consistently, as evidenced by the higher mean reward and reduced standard deviation compared to traditional methods. This indicates that the agent learns a policy that leads to more desirable outcomes on average, contributing to the overall effectiveness of the text generation process.

Considering Figure 3.3, it becomes apparent how the used readability metrics approach a higher mean value over the training run. However, as an example, the Flesch-Kincaid index shows significant improvement over the episodes, other indices such as Dale-Chal do not appear to be influenced by the improved prompts as severely.

Quality vs. Maximum Reward

A notable aspect of the training results is the emphasis on improving the quality of generated samples, rather than simply maximizing the reward. While the maximum reward achieved by the RL-based approach does not surpass that of traditional prompting methods, the consistency and coherence of the text are significantly enhanced, as shown by the significant score stabilization. These results are reflected by both the training process and by the separate evaluation using 300 independent text samples, as the RL approach scores better results in all statistical means except the maximum score.

This observation underscores the importance of considering not only the peak performance but also the distribution of outcomes when evaluating text generation models. While achieving a high maximum reward is desirable, it is equally important to ensure that the generated text maintains a high level of quality across a range of scenarios and inputs. In this regard, the developed approach is successful by consistently generating text that is more readable and contextually appropriate.

Limitations and Opportunities for Improvement

Besides the promising results, several limitations for an extended implementation have to be considered. Firstly, the effectiveness of RL-based text generation may vary depending on the complexity and diversity of the input data and task requirements. This is especially problematic when transferring the approach to an environment where readability does not rely on objective computations but on human considerations and subjective impressions. Furthermore, RL might not be the most suitable solution when dealing with ever-changing environments such as given with user preferences, as training the agent will only be able to cover general aspects of human needs.

Moreover, as this study only considered a discrete word pool and a very specific prompt template making use of the CLEAR-framework, other prompt templates and pools might lead to even better results with improved stabilization rates. It would be particularly interesting to implement the word selection with more flexible approaches, such as frozen LLMs or automatically scraped word pools. Building on this idea in the course of this work, masks should incorporate linguistic schemata which allow the chosen words to be arranged in a human-readable way. Therefore, other mask arrangements will be tested out to explore room for improvement. In this context, it also is desirable to explore different prompt templates using various arrangements of mask spaces, which will then be filled with sentence constructions under linguistic guidelines.

There also might be optimization potential when selecting different LLMs than *Vicuna* for inference operations during training and evaluation. Although Deng *et al.* showed that optimized prompts seem to be transferable to other LLMs, further exploration is needed to understand how different factors influence the performance and generalization capabilities of RL models in text generation.

Additionally, the choice of reward function and evaluation metrics can significantly impact the training process and final outcomes. Readability indices, as used in this work, might lead to environment exploitation by only selecting terms that lead to optimizations such as shortening words and sentences respectively. Future research could explore novel reward functions and evaluation criteria that better capture the nuanced aspects of text quality and relevance, thereby enhancing the overall effectiveness of RL-based text generation systems. In terms of readability indices, future work could focus on how to effectively normalize scores computed by different metrics to allow for a new objective measure that allows for a holistic, more meaningful measure applicable as a reward function for the RL agent.

By incorporating optimizations like this, RL-based text generation systems could potentially achieve more sophisticated reasoning and decision-making capabilities, leading to further improvements in text quality and coherence.

3.4 Improving the Experimental Approach

Although confirming the general applicability of RL in regard to optimizing text quality measured at readability metrics, the study offers various points of improvement as suggested in Section 3.3.6. The four most important points of amelioration will be discussed in the following sections.

3.4.1 Enhancing the Word Pool through Scraping and Clustering

While preliminaries leveraged a manually constructed pool of terms used to fill in the prompt template, an updated approach will focus on structuring the pool more systematically. The first step deals with collecting frequently occurring terms relevant for the task as a base by utilizing existing data corpora. Applying these words as a base, clusters will be synthesized using morphemes as well as corresponding synonym sets.

The initial process starts with *source corpora* which contain single words, specifically structured text information on entire documents. There are two major groups which will be considered:

- **Standard English corpora:** Various text pools with non-fictional content. The vocabulary is adopted in order to represent everyday language use and corresponding vocabulary, such as given in spoken language, magazines, newspapers, or blogs. Fictional texts, such as entire books or subtitles for TV shows as well as movies, are therefore not considered.
- **Task-specific corpora:** Text pools of tasks which are taken from exercises or examinations in school or university context. These pools represent questions which could regularly be asked to LLMs in normal everyday application. Especially, vocabulary related to the scope of organization for a task is overrepresented in this type of data in comparison to normal texts, making this type of text an important addition to the word pool. To reinforce wordings contained in questions and not the corresponding answers irrelevant for the wording of prompts, task pools are preprocessed. The preprocessing implementation is dependent on the respective dataset.

To represent a wide variety of the previously mentioned source corpora, this work aims to utilize multiple pools:

Standard English corpora:

- **Wikipedia Dump**[6]: Regularly updated dumps of all the text content available on Wikipedia.
- **OpenWebText Corpus** [51]: An open-source replication of the WebText dataset used to train GPT-2 (see [52] for reference), derived from web pages shared on the platform Reddit[7].
- **News Commentary Dataset**[8]: A dataset containing a collection of news commentaries on various topics. The dataset moreover focuses on translation tasks from various news articles to other languages, making the data usable for ML translation tasks.

[6] https://en.wikipedia.org/wiki/Wikipedia:Database_download
[7] https://www.reddit.com/
[8] https://opus.nlpl.eu/

Task-specific corpora:

- **SQuAD (Stanford Question Answering Dataset)** [53]: A reading comprehension dataset, consisting of questions posed on a set of Wikipedia articles, where the answer to every question is a segment of text from the corresponding passage.
- **MCTest (Microsoft Research Children's Book Test)** [54]: A public set of stories and associated questions to be used for research on automatic machine comprehension.
- **RACE (ReAding Comprehension from Examinations)** [55]: A large-scale reading comprehension dataset collected from English examinations in China, designed for middle and high school students.
- **ARC (AI2 Reasoning Challenge)** [56]: A challenging dataset of grade-school level multiple-choice science questions.

As the mentioned datasets contain up to multiple millions of separate words, the complexity of items is drastically reduced by applying a strategy of frequency analysis and filtering in terms of POS. Therefore, for each corpus, a standardized *word data corpus* format will be extracted. The process for the creation of each internal corpus follows the same rules: Foremost, the text data will be fully annotated using a POS analysis. As this work aims to only consider meaningful content to be filled in the prompt template mask spaces, only nouns, verbs, and adjectives will be considered in the following process, therefore omitting all other POS, such as conjunctions, prepositions, adverbs, or pronouns. For each of the three groups, a frequency analysis is conducted by counting the occurrences of each unique word. After counting the terms, all words per POS are sorted by their recorded frequency. For the construction of the word corpus, the first n terms — therefore, the most frequent occurrences — are included. Words occurring less are not considered for the final word pool. The amount of words taken per part of speech is a hyperparameter defined as the *word ranking threshold* w_p where p is the considered source corpus. For this work, w will be defined static globally in order to ensure equal cardinalities between all used sets.
In order to focus on words carrying a meaning, certain groups are filtered out in the frequency analysis, namely:

- any special characters mistakenly tagged as specific POS
- forms of the verb "be", "have", "get" and "do"
- words that could be sorted into the wrong type of POS as they are suitable for a number of groups
- weekdays and month names, which tend to appear abnormally often in news articles
- morphemes of the same words represented multiple times within the same ranking

The process of tagging the source corpus and extracting the standardized word data corpus by conducting a frequency analysis on the specified parts of speech is visualized in Figure 3.5.

In another preprocessing step considering all created word data corpora, terms occurring multiple times are deleted so the base word data corpora contain only unique words.

For each word data corpus, separate *word clusters* are put together. Forming the clusters starts with iterating through all words of the given word data corpus. For each unique base term, a new cluster is built recursively using scraped synonyms. Scraping is done using a thesaurus platform which supports assessment of regularity of the term, e.g., whether the synonym is a strong or weak match for the base word which was used for the thesaurus search query. This work will refer to the grade of adequacy as the *matching strength* of a synonym. The matching strength m_w for each scraped

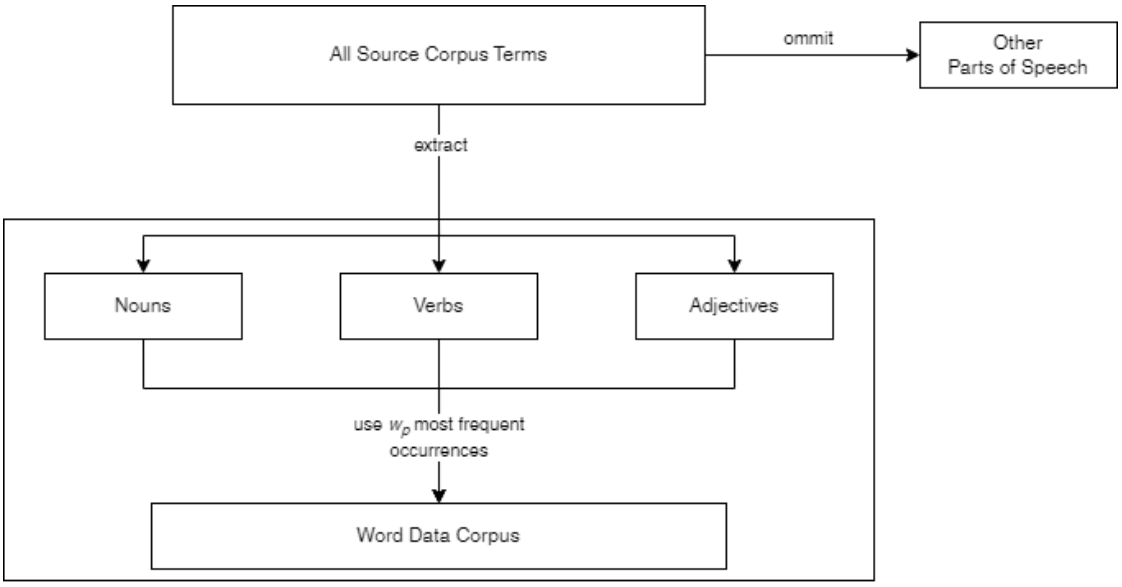

Figure 3.5: Creation of a word data corpus by analyzing a source corpus

term w is saved as a numeric value between 0 and 1. The distribution of values is dependent on the amount of ranks the thesaurus platform provides for synonym sets. For example, when using three categories (strongest match, strong match and weak match) the most common synonyms categorized as "strongest matches" would result in the score of 1 while "weak matches" would equal a matching strength of 0. Categories in between obtain a linearly interpolated outcome, in this case a value of 0.5 for "strong matches". This can be formalized in the following equation:

$$m_w = \frac{\text{rank}(w) - 1}{n - 1} \tag{3.3}$$

where rank(w) is the rank of the term w within the categories from weakest (rank 1) to strongest (rank n), and n is the total number of categories. Saving the matching strength for each scraped base word enables the algorithm to differentiate between commonly and rarely used analogs.

Synonym sub-clusters are limited to a defined recursion depth. A recursion depth level is defined as the amount of hierarchy steps were conducted from a base word in order for a synonym to occur following the scraping process. To avoid cyclic recursion as well as word duplications, two limitations are used for the scraping process:

- Recursion depth is limited to a numeric value. Once a synonym sub-cluster is built and the depth has been reached, no new synonyms are scraped.
- Duplicated base words, which can occur due to a synonym being used for multiple similar words, are not allowed. Therefore, when a new cluster with an already existing term is established, the synonym having a higher matching strength will be chosen, therefore deleting the entry having lower applicability. When the already existing word is a root term taken from the base word corpus, the synonym is instantly ignored. In other words, the original data corpus is immutable and always fully represented in the word pool.

In the end, each single word represented in the pool is analyzed in terms of morphemes. All associated morphemes are annotated with their autonomy, function, or position (see Figure 2.1 for reference) to enable the RL agent to include these characteristics in the action set selection process and to assess their distribution during evaluation.

The basic workflow containing all steps from constructing a new cluster starting with terms selected from a predefined data corpus is visualized in Figure 3.6.

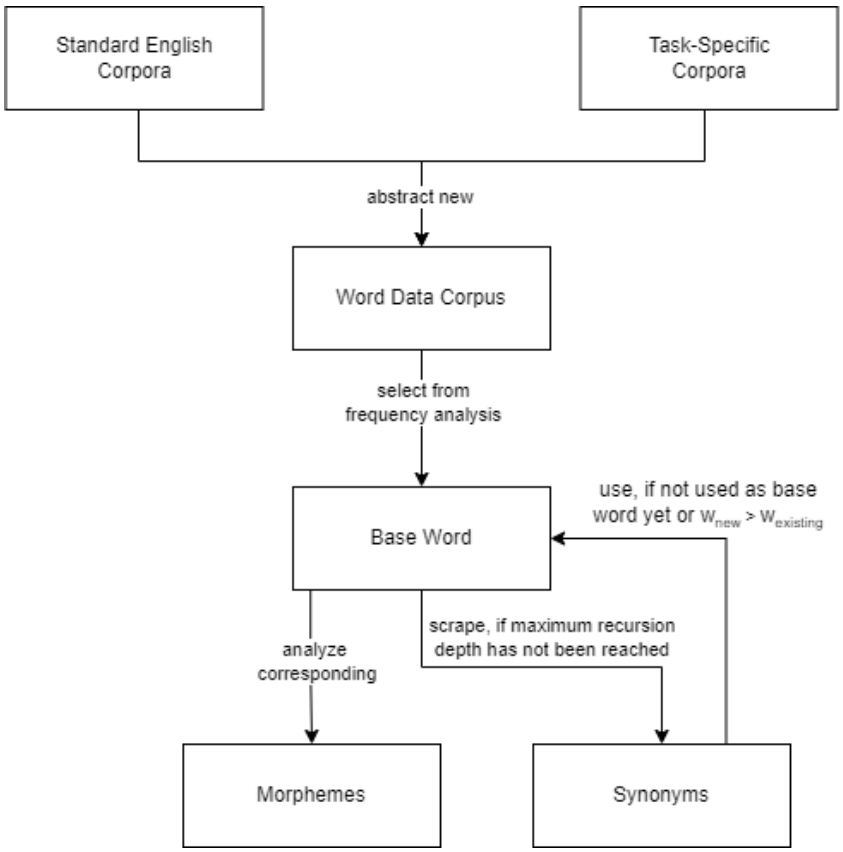

Figure 3.6: General workflow of how the word pool is clustered starting from a given data corpus

The hierarchical relations between the separate components used to construct the word pool are shown in Figure 3.7. Here, scraped synonyms are not visualized as a separate hierarchy component, as each synonym obtained for a base word is handled as a new sub-cluster for the current word data corpus. Moreover, morphemes are not broken down in their internal form of clustering which is used to handle different types of morphemes.

3.4.2 Incorporating Linguistic Fundamentals

Another aspect focuses on embedding linguistic groundwork in the prompt template implementation. While in the preliminaries, as given in Section 3.3.4, prompt templates were defined on a manual, experimental basis following the *CLEAR* framework, enhancements will consider various additional linguistic prerequisites. Based on the experimental approach, the workflow will continue to focus on inserting words from a predefined pool into likewise manually constructed mask templates. However, this time the RL agent will be granted more information in the selection process. This enables the algorithm not only to make decisions on a larger variety of properties, but also to analyze and learn linguistic relationships to select combinations leading to the highest rewards possible.

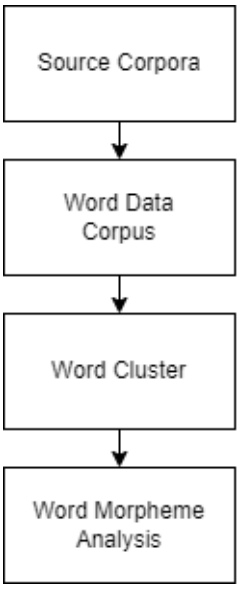

Figure 3.7: Relations between the separate word pool clustering components

Linguistic aspects are primarily represented by the words clustered as given in Section 3.4.1. These clusters will be extended to include a wider variety of linguistic components such as synonyms, antonyms, hypernyms, and hyponyms. By integrating these additional linguistic elements, the RL agent will have access to a richer set of options for generating more diverse and contextually appropriate text.

Utilizing Part-of-Speech Tagging

POS tagging is employed in the clustering process to categorize words in the pool according to their grammatical roles, namely nouns, verbs, adjectives. This categorization helps the RL agent to construct grammatically correct sentences by ensuring that words are used in appropriate contexts. For example, a noun from the word pool could be placed in a position within the sentence where a noun is grammatically required.

By leveraging POS tagging, the RL agent can also learn patterns in sentence structure and improve its ability to generate coherent and well-formed sentences. This approach not only enhances the grammatical accuracy of the generated text but also contributes to its overall readability and fluency.

Template Variability and Sentence Types

To support various sentence types and structures, multiple prompt templates will be designed. Each template will cater to different linguistic requirements that could lead to changes in the content generation and therefore the evaluation unit used to assess text readability. This part of the work is based on the linguistic principles related to prompting presented in Section 2.2.2. Therefore, three major categories are taken into account:

- **Sentence type**: The work of Leidinger *et al.* showed that using different sentence types leads to considerable effects on the LLM output level. [24]

- **Politeness level**: Although politeness only had a minor influence on the average sentence length according to Yin *et al.*, this aspect will be surveyed in order to draw conclusions in terms of readability indices [25]. This could be particularly interesting for the metrics introduced in Section 2.3, as these statistical measures mostly depend on frequency and length analyses.
- **Sentence depth**: This component deals with the complexity the sentences used as the prompt input have. This parameter is included to investigate whether there are performance changes observable in terms of readability when using simpler or more complex sentence structures.

For both groups, three different variations will be included in the template creation process. The three common forms of declarative, interrogative and imperative sentences are used for the considered sentence type. On the politeness level, three separate graduations for the courtesy are included. One variation is used for overly polite or impolite wording. The third neutral level is used to specify the prompt as free of bias as possible, not using any vocabulary contributing to politeness or impoliteness. Sentence depth is also varied on three levels, representing a different degree of complexity each. Using only one main clause conveys the most simple sentence structure possible. Incorporating one additional subordinate clause reflects common spoken and written language structures. The most complex level is represented by three subordinate clauses, leading to longer and more complicated formulations.

Other details on sentence and word level, such as tense and the usage of active and passive aspect, are not taken into account as they did not show major implications in the research of Leidinger *et al.* [24]. However, incorporating all the previously mentioned aspects may help to understand whether the properties are related to each other and influence the overall results in any way when being solely used or even combined.

Improved Mask Spaces

Besides the different types of sentence mutations represented by different templates, the approach also aims to rework the way in which terms are filled in mask spaces. While in the preliminary experiment, each mask served to fill in any word from the predefined word pool followed by possible successors depending on the RL agent's choice, the updated approach differentiates between two types of masks.

The first type focuses on the three grammatical roles assigned during the POS tagging process outlined in Section 3.4.1. Therefore, there are individual POS-specific mask space types for nouns, verbs, and adjectives. The RL agent is only allowed to fill in corresponding terms from the word pool to not violate the grammatical rules defined on the sentence level in the prompt template.

In order to still give the RL agent the opportunity to use the mask spaces to freely select terms represented in the word pool, another mask space type referred to as a *wildcard mask space* is used. This type does not limit the usage of specific POSs. In contrast to the preliminary experiment, each wildcard space only supports the insertion of exactly one term. Therefore, in order to support the generation of entire grammatical structures, a template has to repeatedly use a wildcard mask space.

Using this distinction between POS-specific and general wildcard mask spaces, this study aims to show whether one or the other approach is more suitable for semi-automatic prompt generation techniques.

Prompt Template Rework

The final set of prompt templates is constructed by using the two previously explained technical aspects in a modular combination technique. As for the variations in terms of sentence type, politeness level and sentence depth, each possible combination for the three properties is issued, creating one possible template approach each. This leads to $3^3 = 27$ possible unique combinations.

Each combination is implemented using two concrete prompt templates. One template utilizes only POS-specific mask spaces, while the other template focuses on wildcard mask spaces exclusively. Overall, there are $27 * 2 = 54$ prompt templates used for the updated methodology. The components of the modular approach are visualized in Figure 3.8.

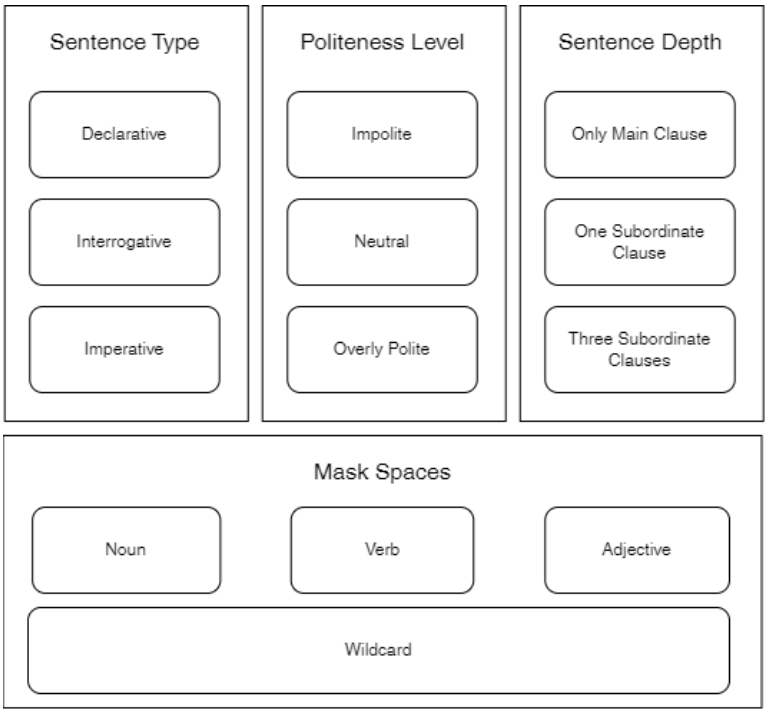

Figure 3.8: Modular components considered for the combination of different prompt templates

Each template additionally specifies exactly one space where the input term for the topic which is handled is inserted. Just like in the preliminaries, the input determines the topic which is contextually used for the generation process. The selection process for the input topic is outlined in Section 3.4.4.

3.4.3 Normalizing Readability Metrics

In the revised implementation, the design of readability metrics used as the RL reward function is reworked as well. Although the test of four used metrics showed promising results in the experiment, the variation in their individual deviations shows that an enhanced normalization is required.

To represent this normalization, each metric will be run through a *normalization process* manually deduced from the distribution of score value ranges as given in corresponding statistical models. Therefore, the normalized score for each applied metric is separately calculated using a custom normalization function n:

$$s_m = n(m) \tag{3.4}$$

where each metric m is processed through a normalization function n, resulting in the normalized signed score s $(-1 \leq s \leq 1)$.

The mapping is done by using score anchor points described by originators or advanced statistic research. For example, when the metric score represents the school class level a text is suitable for, mapping would be linearly interpolated with the highest school level and therefore the highest text complexity resulting in the normalization score -1 and the lowest class level with the lowest complexity to normalization score 1. Table 3.2 summarizes the three most significant value ranges provided by each metric, along with the general interpretation of the corresponding scoring.

For the normalization function $n(m)$, value termination is done using three predefined values for the easiest and hardest score, as well as a neutral value. Values below the easiest magnitude n_1 map to score 1, the neutral score n_2 to 0 and values above the hardest index n_3 to -1. Moreover, a parameter d is introduced indicating the score direction of each metric following the regulation:

- $d = 1$ if higher scores indicate more difficulty.
- $d = -1$ if higher scores indicate more ease.

Parameter d can be generally deduced from the range values following:

$$d = \begin{cases} 1 & \text{if } n_1 < n_2, \\ -1 & \text{if } n_1 > n_2. \end{cases}$$

Values in between ranges are interpolated linearly, resulting in the following definition for the normalization function n for a given metric m:

$$n(m) = \begin{cases} 1 & \text{if } d \cdot x \leq d \cdot n_1, \\ \frac{d \cdot (n - n_1)}{d \cdot (n_2 - n_1)} & \text{if } d \cdot n_1 < d \cdot x \leq d \cdot n_2, \\ \frac{d \cdot (n - n_2)}{d \cdot (n_3 - n_2)} & \text{if } d \cdot n_2 < d \cdot x \leq d \cdot n_3, \\ -1 & \text{if } d \cdot n > d \cdot n_3. \end{cases} \tag{3.5}$$

Table 3.2: Value ranges and interpretations for readability metrics

Readability Metric	Range	Interpretation
FK [31]	90-100	Very easy texts, easily understandable by an average 11-year-old student.
	60-70	Easy to understand by 13- to 15-year-old students.
	0-30	Very difficult texts, best comprehended by university graduates.
CLI [32]	1-4	Very easy texts suitable for primary school level.
	5-8	Intermediate texts suitable for middle school level.
	9-12+	High difficulty texts suitable for high school and above. Post-graduate level complexity at the highest scores.
DC [33]	4.9	Very easy texts, easily understood by 4th graders.
	5.0-5.9	Easy texts, understandable by 5th or 6th graders.
	9.0-9.9+	Difficult texts, understandable by 13th to 15th graders.
GF [34]	6	Easy texts, readable by 6th graders.
	10	Intermediate texts, readable by high school sophomores.
	17+	Very difficult texts, post-graduate level complexity.
$SMOG$ [35]	1-6	Very easy texts, suitable for younger readers.
	7-12	Intermediate texts, suitable for middle and high school levels.
	13-18+	Very complex texts, requiring advanced education.
ARI [36]	-5 to 0	Very easy texts, suitable for early primary school levels.
	1-8	Intermediate texts, suitable for lower and middle grades.
	9-14+	Difficult texts, suitable for high school students and above.
LIX [37]	20-30	Very easy texts, like children's books.
	30-40	Easy texts, such as popular fiction books.
	50-56+	Difficult texts, such as academic papers or legal documents.

As for the definition of the anchor values, ranges given in table 3.2 are used. n_1 will map to the easiest value possible, while n_3 is assigned as the hardest value. n_2 is calculated using the mean score of the neutral range. d is set individually depending on whether the ease indicated is scaled positively or negatively. All parameter sets used to define the normalization function $n(m)$ for each considered metric are summarized in table 3.3.

3.4.4 Adjusting the Reinforcement Learning Approach

Due to the changed dimension of action space and environment, the RL algorithm along with the logic used to pass in the topic for generated texts needs to be adjusted. While the overall objective is to adapt to the modifications presented in this section so far, another sub-goal is to improve the training process used for the preliminary experiment as well.

Table 3.3: Parameter sets for readability function $n(m)$ for each metric

	n_1	n_2	n_3	d
FK	100	65	0	-1
CLI	1	6.5	12	1
DC	4.9	5.5	9.9	1
GF	6	10	17	1
$SMOG$	1	9.5	18	1
ARI	-5	4.5	18	1
LIX	20	35	56	1

Contextual Topic Selection

The topic that determines the general content of the generated text is the only input dimension used for the RL approach. While topic selection was randomized with a few pre-selected terms in the preliminaries, the updated approach adds contextual information to the selectable inputs. This procedure enables the agent to differentiate prompt generation depending on the context of the topic used as input. As a consequence, variety for the optimization process is established, forming an important difference to the preliminary experiment.

To reduce the problem dimension, context is not established using complex representations such as word vector embeddings. Instead, there is a static set of *topic groups*. Each group contains a list of five input terms. For this work, the following groups with corresponding input words have been defined following WordNet hyponym search [29]. For each category, only terms undoubtedly providing information for the specified context are selected. For this work, the following topic groups will be used:

- **Technology**: computer, internet, software, hardware, AI
- **Sports**: soccer, basketball, tennis, baseball, Olympics
- **History**: Egyptian Age, Roman Empire, French Revolution, Second World War, Cold War
- **Personalities**: Jesus, Christoph Kolumbus, Charles Darwin, Isaac Newton, J.F. Kennedy
- **Environment**: ecosystem, biodiversity, conservation, pollution, sustainability
- **Health**: nutrition, exercise, plague, cancer, Coronavirus
- **Countries**: USA, Germany, Egypt, China, Australia

General Reinforcement Learning Adjustments

To grant the agent more flexibility in the context of term selection and the action insertion in pre-defined mask spaces, the RL prompt generation process is split into two separate steps. This procedure aims to break down the prompt creation between the selection of the template and the words filled in from the word pool:

1. **Prompt template selection**: The agent selects the prompt template which is used as the base for the LLM prompt input. In this initial step, mask templates have not been filled yet.
2. **Mask fill**: The prompt template selected in the first step is passed to another RL step that targets to fill in words into the available mask spaces.

Like in the preliminary RL approach, the algorithm is represented as a MDP in its simplified version. However, there are major differences in the characterization of the environment as well as the available actions the agents can select from. In the updated approach, the environment is modeled

with a more comprehensive representation that incorporates the enhancements discussed in Section 3.4.1 and Section 3.4.2 Specifically, the environment consists of an enriched word pool created through scraping and clustering, as well as the integration of linguistic fundamentals along with the updated readability score calculation for reward determination.

The environment in the updated MDP framework is significantly enhanced through the systematic structuring of the word pool and the incorporation of linguistic elements. For the RL algorithm, these aspects are included in the DQN environment representation. In the following, the separation of the RL system is explained by outlining the prompt template and mask filling approaches.

In the first step, the agent will determine a prompt template that is used for the text generation. As outlined in Section 3.4.4, the contextual classification for the input topic is only one-dimensional, providing exactly one class for the used term. This classification is passed to the RL agent as environment input data. For the selection process, on the output layer the underlying sentence type, politeness level and sentence depth (see Section 3.4.2 for reference) are encoded using unique, numerical indices whereas the mask space type (wildcard or pre-defined POS) is represented with a boolean flag. The agent thus determines which attributes the desired prompt template should have, providing output of a four value classification tuple. Each unique attribute combination points to one of the templates as defined in Section 3.4.2.

The action space for the updated template filling agent is also more sophisticated. The mask filling process differs depending on whether the template uses pre-defined POS or wildcard spaces, but follows the same procedure in both cases. An action selection sequence is repeated iteratively for each mask available in the selected template:

1. **Pass input parameters to the agent**: The environment input for this part of the RL system consists of two parts: Prompt template information and filling interim status.
    - Encoded information data for the selected topic group and the prompt template chosen in the first step get passed using numerical indices, just like previously laid out according to the first part of the approach. Lastly, the type of mask space is passed, providing four possible choices (noun, verb, adjective, or wildcard space).
    - To allow for the purposeful selection of rewarding word combinations, for the interim status information on the previous iteration steps is passed, in this case on the already selected words. This is done by using a reserved amount of input parameters in the DQN architecture that are only filled when information on previously inserted words is present. The maximum amount of mask spaces fillable by the agent is a global hyperparameter that has to be accounted in the prompt template definition as well. Using these properties, the agent is enabled to link word significations with arrangements issued for the selected prompt template.
2. **Action selection**: For the final step, the word pool clustered as given in Section 3.4.1 is used as the action space for the agent. However, the usage of the pool is restricted depending on the type of mask space that is currently iterated:
    - When the mask space is POS-tagged, only clusters belonging to the given type of POS are available for choosing.
    - When the mask space is a wildcard space, the entire pool can be used.

After all words for the template have been chosen for the available mask spaces, the entire prompt can be assembled and passed to the used LLM instance. The resulting text is then passed to the evaluation unit utilizing normalized readability metrics to return the reward signal. This signal is used to update both parts involved in the RL process, thus adjusting DQN weights to encourage positive compositions.
Just like in the preliminary experiment, agents' policies employ an epsilon-greedy strategy, balancing exploration and exploitation. This strategy ensures that the agent efficiently learns to select terms that maximize the reward based on readability metrics.

The reward function in the updated approach is designed to capture the nuanced aspects of text readability. As mentioned in Section 3.4.3, readability metrics are normalized to provide consistent reward signals. The reward function considers the normalized scores from all mentioned metrics. This refined reward structure ensures that both parts of the agent focus on generating text that has enhanced readability. The reward signal is passed to both RL selection processes previously mentioned. This process enables the system to be trained to find the perfect combination between a fitting template as well as the insertions filled into corresponding mask templates. Only optimizing one of the two aspects is not the aim of this work, therefore excluding separate reward signals for both parts.

The interaction between all RL components, starting with the topic input and ending with a fully fledged prompt template that is passed to a LLM deducing the reward using normalized readability metrics, is shown in Figure 3.9. The shown features represent the entire workflow of all the elements explained in the rundown of the updated approach.

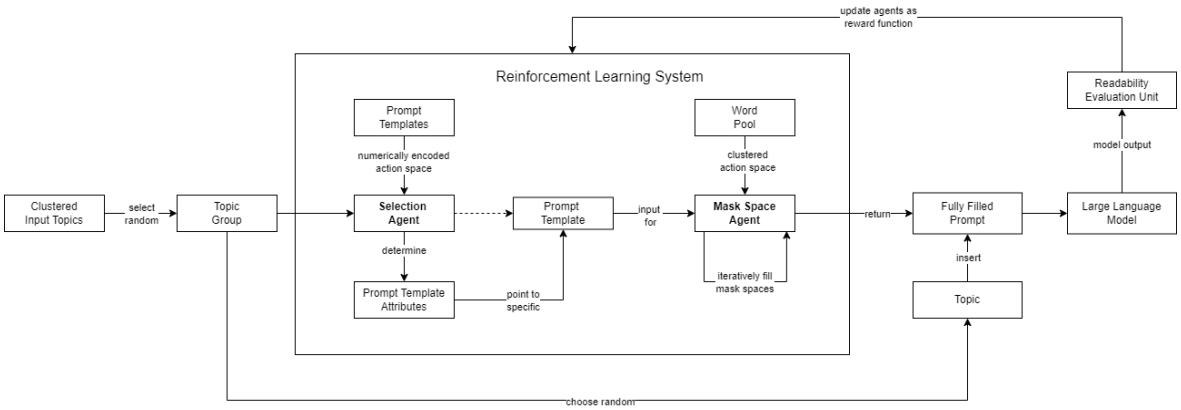

Figure 3.9: Systematic outline of the RL system used to determine a full prompt template from an input topic

Stabilizing Training using a Target Network

As suggested by Mnih *et al.*, the initial naive ML-based approach can be fine-tuned using a target network to enhance training stability [57].

The target network, a separate neural network with fixed parameters, plays a crucial role in stabilizing the training by providing more stable and consistent target Q-values for the DQN to approximate. Instead of directly using the DQN's Q-value estimates as targets for updating the Q-network parameters, the target network periodically copies the weights of the DQN, ensuring that the target Q-values remained temporally consistent over several iterations. During training, the target network

was periodically updated to match the parameters of the DQN at static intervals. This decoupling of the target Q-values from the current policy helped to reduce the variance in the updates and provides a more reliable signal for guiding the agent's learning process. [38, pp. 147-149]

To verify whether this technique is also suitable in the context of using deep RL for readability optimization, the experimental approach designed in Section 3.3 was extended with a target network for training.

In implementation, the target network was updated every 10 episodes to stabilize training and improve convergence. The results, using the same visualization and smoothing method as Figure 3.2, are visualized in Figure 3.10. The statistical base data of both training processes is summarized in table 3.4.

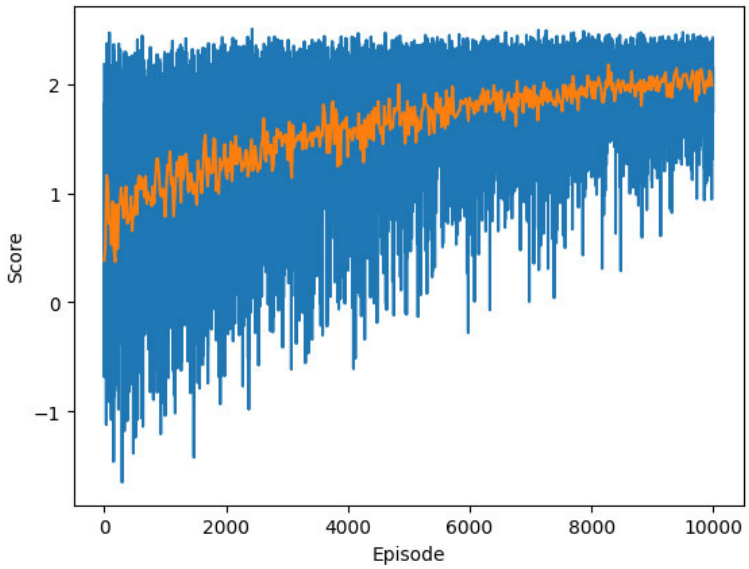

Figure 3.10: Score distribution over a training run of 10,000 episodes: In this run, the agent's model was trained with a target network

Table 3.4: Comparison of statistical data between the training without and with a target network used in the agent's network

	Mean	25%	Median	75%	St. Deviation	Min	Max
No target network	1.16	0.64	1.26	1.81	0.84	-2.35	2.67
Target network	1.61	1.29	1.75	2.11	0.64	-1.65	2.51

The introduction of a target network significantly enhances the stability of the training process and mitigates issues such as overestimation bias and variances in Q-value estimates, leading to smoother convergence and more reliable performance. This interpretation is especially evident in the 76% difference between the standard deviation of 0.84 without using a target network and 0.64 with the stabilization mechanism (see Table 3.4 for reference).
While both mean and median of achieved scores is higher when using a target network, it is particularly clear that the noise within the rewards is significantly lower in the second training run visualized in Figure 3.10. As a result, the RL-based text generation system trained with a target network demonstrates improved learning efficiency and robustness, achieving higher average rewards over the training period in the initial experimental setup.

These findings suggest that using a target network in the improved implementation should be set up in the DQN architecture. This way, stability should be enhanced significantly in the updated approach.

3.5 Expanding Hypotheses

Based on the variables that were chosen for the updated RL methodology, new hypotheses are formulated. Each hypothesis aims to verify or falsify certain correlations between used features and the change of normalized readability evaluation.

The following hypotheses are put forward in regard to impact of RL and word selection on text readability:

1. H_2**:** *RL improves text readability over training time in the updated implementation approach.*
   Approach: Compare readability scores of generated texts from early and late training episodes. Define two groups, E_1 (early episodes) and E_2 (late episodes), and compare their normalized readability scores using the readability metrics applied in this work.
2. H_3**:** *Matching strength selection of words significantly influences readability.*
   Approach: Group episodes by matching strength values and analyze mean readability scores across these groups to identify correlations.
3. H_4**:** *Recursion depth in word selection significantly affects readability.*
   Approach: Compare readability scores across different recursion depth levels to determine the impact on text readability.
4. H_5**:** *Part of speech of selected words significantly impacts readability.*
   Approach: Categorize episodes based on the dominant part of speech used and compare readability scores for different categories.
5. H_6**:** *Morpheme types used in word selection significantly affect readability.*
   Approach: Group episodes by types of morphemes used and evaluate the differences in readability scores.

The following hypotheses are put forward in regard to impact of template selection on text readability:

1. H_7**:** *Sentence depth in templates significantly impacts readability.*
   Approach: Compare readability scores across templates with varying sentence depths to assess their impact.
2. H_8**:** *Politeness level in templates significantly affects readability.*
   Approach: Group episodes by politeness level used in templates and compare readability scores across these groups.
3. H_9**:** *Sentence type in templates significantly influences readability.*
   Approach: Analyze readability scores based on different sentence types used in templates to determine their effects.
4. H_{10}**:** *There is a significant difference between results of POS-specific masks and wildcard masks.*
   Approach: Compare readability scores for episodes using POS-specific mask spaces versus those using wildcard mask spaces to evaluate effectiveness differences.

4 Implementation and Results

This chapter presents the updated implementation details and results obtained from conducting the optimized RL-based text generation system training. Details about the chosen frameworks and methods, along with a summary of the hyperparameter setups used in the training process, are presented.

Foremost, an outline of the software architecture is given. One major difference to the preliminary experiment, which was implemented using a simple monolithic architecture setup, is the usage of a part-wise distributed setup. The system leverages multiple components to represent the sophisticated, reworked RL training and inference environment. At its core, the architecture used a central controlling unit which initiates the entire model training and supports access to data collected during the training process. This controller can be seen as a *modulithic* system providing all hyperparameter sets for training as well as prompt templates and corresponding topic groups. Other components of the implementation ecosystem are outsourced in microservices. These services are then linked in order to depict the entire updates methodology setup as given in Section 3.4.

This reworked implementation approach is used to fulfill three important goals:

- **Enhance maintainability while keeping centralized control**: Distributing various tasks over microservices makes it easier to identify sources of error and bottlenecks within the entire application. However, keeping the overall control in a monolithic manner makes sure that all components can work together in a centralized way. This supports training and inference experiments to be coordinated more controlled.
- **Support asynchronous training and evaluation procedure**: Preliminaries showed that the training, which required interaction between the DQN training, evaluation steps and LLM inference, are time-consuming when issued synchronously. Thus, splitting up the procedure into microservices independently interacting enables the training to run asynchronously. This does not only speed up training times but also allows individual components to define data-wise dependencies which are then used to optimize resource flows.
- **Extensibility and exchangeability**: By providing unified interfaces for the given microservices, the system can be extended easily. This enables future research not only to extend existing software implementations following this work, but also to develop new solutions with modified components, as long as interface data exchange formats are followed.

The general architecture setup used is visualized in Figure 4.1. Arrows signalize interaction between the individual components, establishing a flow of data. The *learning environment controller* resembles the modulithic unit responsible for coordinating microservice distribution and initialization of the RL training process. The *word pool service* not only initializes the analysis and scraping process to build the word pool, but also serves the entire pool representations, prompt templates and topic groups with their items. To evaluate LLM-generates texts, the *readability evaluation unit service* calculates raw and normalized readability metrics as described in Section 3.4.3. Training is conducted using the *learning agent service*, which saves all data for initialization and individual training episodes in the *training documentation service*. Lastly, saved models can be accessed using the *prompt inference service*, allowing for a fully automated prompt provision using only topic information as input.

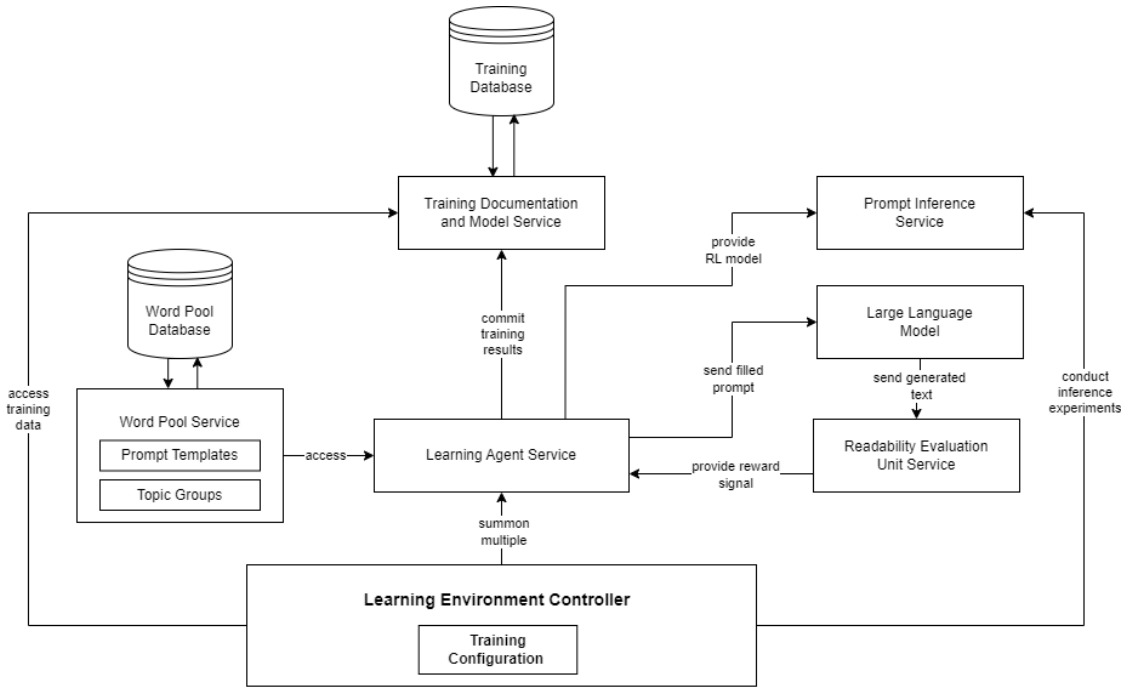

Figure 4.1: Rough software outline for the applied architecture

Communication between individual components will be issued through web requests sent using HTTP. Parts of the system thus know which data standards have to be fulfilled when transmitting requests and which data can be expected when receiving responses from corresponding counterparts. Using HTTP communication within a modulith offers several significant advantages that align well with both practical needs and scientific principles. In the context of the modulith given in this work, HTTP communication supports modularity and clear separation of tasks. Each component can expose its functionality through interfaces, allowing other parts of the system to interact with it as a service. This modular approach simplifies development and maintenance, as components can be developed and tested independently. Furthermore, the use of HTTP ensures that these interactions follow RESTful principles, promoting statelessness and uniform interfaces, which contribute to the overall robustness and simplicity of the system. The choice of HTTP also allows future scalability and system evolution. As the system grows, individual components that communicate via HTTP can be more easily transitioned into independent microservices if needed. This potential for gradual evolution without significant rework aligns with long-term architectural goals. Additionally, HTTP's universal nature makes it easier to interact with external systems, enhancing the system's interoperability and extending its effectiveness beyond the scope of the exemplary deployment used in this work.

In the following, this chapter will present implementation details for the general project setup and all important named components used for the updated software architecture. In the end, interaction between the individual parts will be employed to conduct training and evaluation inference in order to collect result data for this work.

4.1 Project Setup

Like for the preliminary experiment, the programming language Python has been chosen for implementation of the various needed software components. For the core RL environment, the main reason for the decision is the established quality of ML algorithms that has been put to use in the preliminaries already.

For the microservice components, Python presents various solutions that can be leveraged to expose ordinarily language-exclusive features using versatile interfaces.
For this work, the framework *Flask* will be used for all microservice requirements. This lightweight tool, based on *Werkzeug* and *Jinja2*, offers a straightforward and efficient way to create and manage web applications. Flask supports the provision of database schemas and facilitates connections and interactions with databases, providing maximum flexibility by supporting both SQL and NoSQL databases. This flexibility is essential for handling the diverse data storage needs that arise in the RL-based text generation system. [58, p. 1, pp. 27-28]

Using Python as the general solution for implementation allows this project to be shipped as an all-in-one solution. This is advantageous because Python's extensive standard library and ecosystem of third-party packages, such as Flask, provide support for a wide range of functionalities needed for the application. The modulithic core can directly take advantage of the interface definitions within the same project, ensuring seamless integration and reducing the overhead of context switching between different programming environments.

4.2 Word Pool Service

The *word pool service* is used to provide the entire word dataset that is obtained using the scraping and clustering procedures detailed in Section 3.4.1. Additionally, the collection of topic groups with context items (see Section 3.4.4 for reference) and all prompt templates (see Section 2.2.2 for reference) are served using this service. This allows the training to adjust dynamically when changes to the underlying data are made.

In the following, this section describes the data representation setup as well as the procedure used to scrape and represent the word pool data to make them usable for the RL agent training process.

4.2.1 Word Pool Database Setup

The data representation for the updated word pool is far more complex than in the preliminary approach. Therefore, instead of relying on simple textual file formats, a relational SQL database system is used to establish linkages between different objects that are represented as entities within the database. The relational model does not only allow efficient storage and retrieval of data but also supports complex queries and transactions being essential for the integrity and consistency of the dataset.

Figure 4.2 shows the relational table schema in an entity-relationship diagram. This diagram essentially depicts the hierarchical relations visualized in Figure 3.6 and Figure 3.7. The diagram includes entities such as *words*, *synonyms*, *morphemes*, and their interrelations, showcasing how each word

entry is connected to its synonyms and morphemes. Moreover, recursive scraping of synonyms is supported, creating a metadata object which serializes the current scraping recursion depth, the synonym matching strength as well as a reference to the base word which was used to extract the corresponding synonym. Additionally, the schema illustrates how the database supports various linguistic properties such as POS and their classifications, enabling the RL agent to access and utilize the entire set of linguistic data for training.

The chosen data models allow structuring the connections between words and their associated linguistic features, which are important for the performance of the RL agent. By organizing the data in this manner, the system ensures that each component of the word pool can be efficiently set during scraping and clustering and accessed during the training and inference processes. This setup does not only enhance the maintainability and scalability of the database but also supports the complex interactions required for prompt generation.

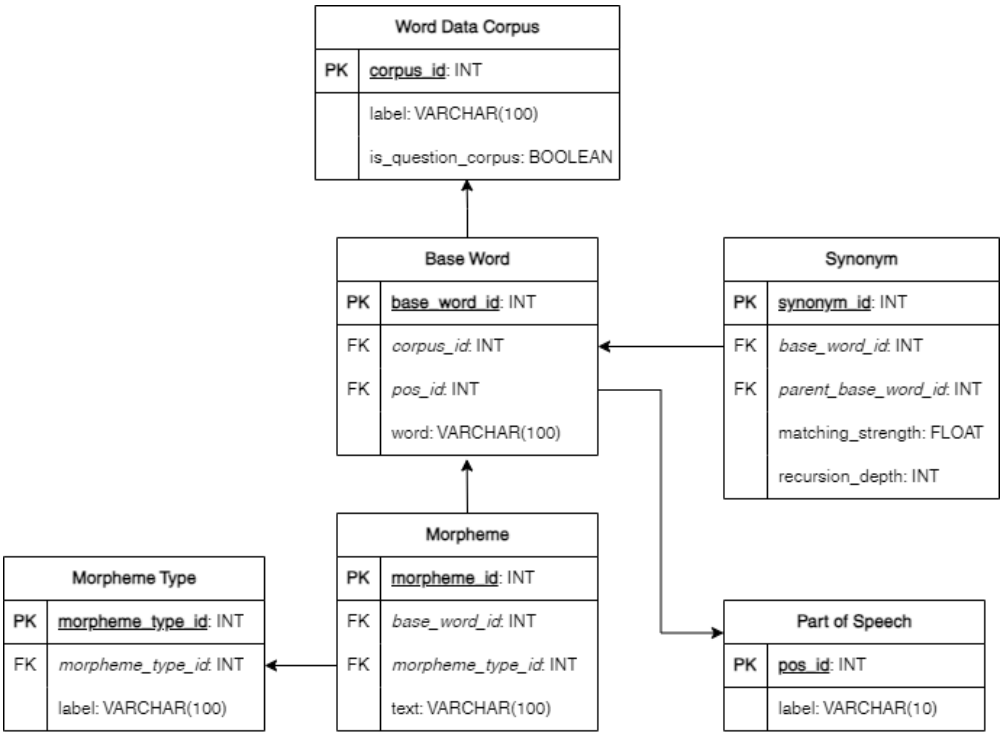

Figure 4.2: Entity-relationship diagram for the word pool database setup

Another separate data model is used to represent the prompt templates along with their properties used to identify individual templates with mask spaces in the RL selection process. The template entities, as shown in Figure 4.3, only serialize the template text where mask spaces are defined using pre-defined characters substituted by the RL algorithm. The properties represented by the template are saved using the associated table references for property entities.

Lastly, topic groups are also represented within the word pool service. Therefore, the separate topic groups along with their individual entries described in Section 3.4.4 are saved as database serialization as well. The corresponding visualization diagram is shown in Figure 4.4.

Matching the given concepts, PostgreSQL is selected as a relational, SQL Database Managament System (DBMS) allowing for efficient data storage, complex querying capabilities, and transactional support. This choice ensures robustness and scalability, making it suitable for handling the intricate

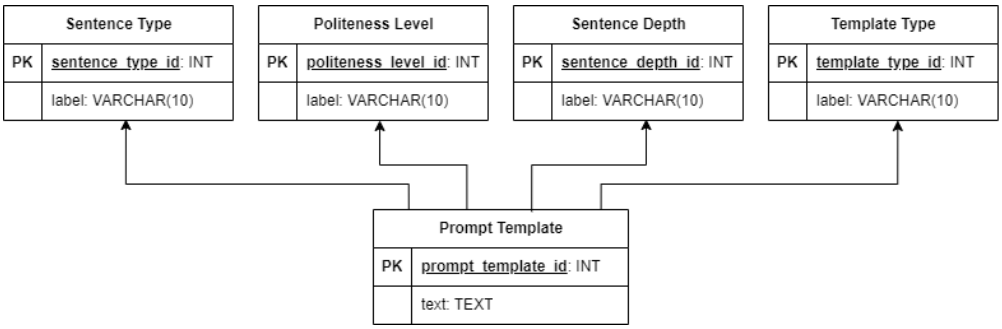

Figure 4.3: Entity-relationship diagram for the prompt template database setup

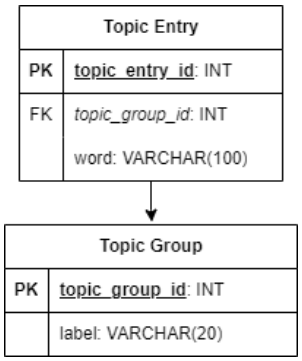

Figure 4.4: Entity-relationship diagram for the topic groups database setup

relationships and large datasets involved in the project. Additionally, Flask supports database model integration for PostgreSQL flawlessly using *SQLAlchemyAdapter*, which facilitates Object-Relational Mapping (ORM). This integration simplifies database interactions by allowing to use Python code to manage database records, perform queries, and handle migrations. The combination of PostgreSQL and SQLAlchemy within the Flask framework ensures a highly efficient and maintainable codebase, streamlining the development process and enhancing the overall system performance. [59]

However, in an exemplary production environment, a dynamic flask configuration allows selecting different DBMS for database setup. For example, using a local SQLite DBMS setup would allow the system to run without additional external database dependencies.

4.2.2 Scraping and Clustering Implementation

The database filling process firstly requires the preparation of all text corpora enumerated in Section 3.4.1. All data sources are downloaded as text files and then included in the local microservice project to simplify the preprocessing and text extraction steps.

To simplify the following analysis steps, all text data are required to be present as raw text data. As each corpus used for this work is presented in a different data structure originally, preprocessing steps differ:

- **Wikipedia Dump**: To reduce data complexity, the Wikipedia dump used for this work is limited to the *Simple English Wikipedia* torrent. The download is given in XML file format, thus preprocessing is used to reduce data overhead which results from this form of representation. Using the *Python XML ElementTree API*[9], relevant content is written into a separate text file.
- **OpenWebText Corpus**: As the entire corpus is extremely large, only the very first subset as given by the dataset creators is used for this work. Data is extracted from multiple tarfile-archives with many text content files, which are aggregated into a single, central text source during preprocessing. This subset itself serves more than 300 million words.
- **News Commentary Dataset**: News datasets translating from German to English were used for this work, resulting in a raw data amount of about 25 million words.
- **SQuAD**: All questions with their answers are serialized in a single JSON file. All questions are extracted in a separate text file by reading the specific fields from the JSON objects, resulting in about 130,000 question entries.
- **MCTest**: Articles, questions and answers are split into several text files for ML training and test splits using a static schema. Questions are extracted using a regular expression in all relevant files, once again aggregating results in a separate text file.
- **RACE**: Questions, corresponding articles and answer options are split into various JSON files. Moreover, there is already a preset for splitting data in training, validation and test steps. Therefore, preprocessing involves recursively visiting all folders of the dataset and processing each JSON file separately. In the end, all questions are written into a single aggregation text file.
- **ARC**: Texts are directly given in a corpus file, which is immediately used for the word extraction process.

After the raw text data for each corpus has been loaded successfully, the POS annotation step is initiated. Using the NLP package NLTK[10] for Python, one text corpus is processed at a time. To reduce memory usage, the frequency analysis is not conducted in a single step. Instead, texts are processed line-wise, allowing the system to regularly write counting results in a file before creating the final ranking of frequency occurrences. To finalize the word extraction, each group of POS (nouns, verbs, and adjectives) is analyzed in terms of word occurrence quantity, forming frequency rankings as described in Section 3.4.1. To do so, each word with its amount of occurrences in the corpus is saved in a separate text file and grouped by POS. To create the ranking needed for the base word selection, the occurrences are sorted in a descended order. For this work, the maximum amount of frequently occurring words, defined as the *word ranking threshold* w_p, is set to 10. Using a relatively low value like this allows all words obtained from the scraping process, which is explained in the following, to have a more meaningful standing in the RL training process. This decision follows research from Deng *et al.*, especially accounting performance increases resulting from the usage of rarely used word synonyms.

Database models, following the relations given in Figure 4.2, are modelled using a SQLAlchemy database setup. Base words for the corpora are directly inserted using their corresponding POS in the initialization progress, resulting in a base pool of 30 words for each corpus. The initial collection of unique words, filtering duplicates, obtained from rankings of individual corpora along with their assigned POS is shown in Appendix C. Overall, 209 unique base words are used.

[9] https://docs.python.org/3/library/xml.etree.elementtree.html
[10] https://www.nltk.org/

Using these terms as a starting point, the scraping process collecting thesaurus synonyms is initiated. Using the public Thesaurus[11] website, a separate scraping procedure is conducted for each item of the base word set. Sending a website request using the Requests[12] package, BeautifulSoup[13] is used to analyze the page structure in order to extract relevant information, namely the corresponding synonyms and their matching strength which is determined by the category used on the webpage:

- **Strongest Match**: $rank(w) = 3, m_w = 1$
- **Strong Match**: $rank(w) = 2, m_w = 0.5$
- **Weak Match**: $rank(w) = 1, m_w = 0$

To maintain word context, only synonyms consisting of a single term are used, thus omitting phrases that are composed of multiple words. This way, new base words pointing to their structural word parent are inserted in the database. This process, only operating on one level of depth for the scraping process, 2680 synonyms, resulting in an overall count of 2926 words available for the RL agent action space.

The last step for the full word pool construction is morpheme analysis for the collection of base words. Analysis is conducted using the package morphemes[14], thus disassembling each base word into separate morphemes along with their corresponding morpheme type. Morpheme types for this work are, just like in the package implementation, differentiated into root, bound, free, prefix and suffix morphemes.

4.2.3 Data Access and Mutation Interface

All database write operations that are laid out to either retrospectively extend or shrink the word pool are exposed using a Flask-based web service. For the scope of this work, however, these identical interfaces are as well leveraged to carry out all data access operations, allowing the RL agent to retrieve all information necessary for its action space.

The most important type of service for this work is the operation used to obtain the word pool data serving as the agent's action space. To obtain all words in their hierarchical order per analyzed corpus, a simple GET request is served:

```
curl http://localhost:5556/api/wordpool
```

The response provides a list of all separate word extractions, divided by their source corpora. If there are some, each base word contains a separate list of their synonym base words along with their matching strength. As a recursion depth limit of one has been set for this work, extracted synonyms do not have another layer of corresponding synonyms. Moreover, for each base word, the morpheme analysis is serialized in the response as well, enumerating all extracted morpheme texts along with their type. Listing 4.1 shows an example of one base word with one exemplary synonym entry. For both base words, the broken down morphemes can be accessed as well.

[11] https://www.thesaurus.com/
[12] https://pypi.org/project/requests/
[13] https://pypi.org/project/beautifulsoup4/
[14] https://pypi.org/project/morphemes/

Listing 4.1: Output for one base word with one examplary synonm and full morpheme annotation


```
{
    "id": 89,
    "word": "much"
    "morphemes": [
        {
            "morpheme_type": "root",
            "text": "much"
        }
    ],
    "part_of_speech": "adjective",
    "synonyms": [
        {
        "matching_strength": 0.5,
        "recursion_depth": 1,
        "word": {
            "id": 2161,
            "morphemes": [
                {
                    "morpheme_type": "root",
                    "text": "abound"
                },
                {
                    "morpheme_type": "bound",
                    "text": "ance"
                }
            ],
            "part_of_speech": "noun",
            "word": "abundance"
        }
    ]
}
```


To simplify ML procedures in which numerical representations for actions and their attributes, another app route only returns unique identifiers for base word objects and their associated morphemes. These values, along with numerical properties, do point directly to corresponding primary keys of database objects. An exemplary route for numerical representation is fetched using a GET request like this:

```
curl http://localhost:5556/api/wordpool/id
```

An output example for this interface, representing the same base word with one example synonym as Listing 4.1, is shown in Listing 4.2.

Listing 4.2: Output for one base word in the numerical form of representation


```
{
    "id": 89,
    "morphemes": [
        {
            "id": 85,
            "morpheme_type": 1
        }
    ],
    "part_of_speech": 3,
    "synonyms": [
        {
            "matching_strength": 0.5,
            "recursion_depth": 1,
            "id": 2161,
            "morphemes": [
                {
                  "id": 2892,
                  "morpheme_type": 1
                },
                {
                  "id": 2893,
                  "morpheme_type": 2
                }
              ],
            "part_of_speech": 1
        }
    ]
}
```


As these encodings are not suitable for inference operation which utilize only corresponding words, another decoding app route enables to retrieve only the word for a given identifier. The route requires explicit specification of the identifier that is used for the base word. An example to fetch word data for the base word using the identifier *89* could look like this:

```
curl http://localhost:5556/api/wordpool/id/89
```

The response body for a request like this returns a JSON object with a single property representing the corresponding word that can then be used to fill a mask in the prompt template. Listing 4.3 shows an example response from this endpoint.

Listing 4.3: Output for one base word as text representation


```
{
  "word": "much"
}
```


In addition to the interfaces used for plain word pool data access, POST and DELETE endpoints are provided to allow adding new words or to delete existing objects. The POST operation is used to insert a new word independently of other base corpora. As no scraping is conducted when inserting a new word using this interface, potential synonyms are not accounted. On the other hand,

operations for POS determination and morpheme analysis are automatically started, and database model primary keys are generated by default, thus the POST request body must only contain the JSON object with the new word as given in Listing 4.3.

The word deletion endpoint requires specification of a word using its primary key identifier as reference. When deleting a base word, all synonyms scraped from the word are deleted in a cascading manner. In the background, morpheme and POS information objects are omitted as well. The same endpoint URI as for referencing a single base word using its identifier is used for this process, with the only difference being the specification for the HTTP DELETE operation.

Besides the word pool itself, the described service also provides data access to the prompt templates which can then be filled by the RL agent. Following the database model implementation according the schemas given in Figure 4.3, two Flask endpoints are provided for the templates. The first route returns a raw textual representation for all created templates along with their properties, providing a handy way to output the prompt template selection in an easily interpratable way. An exemplary response only containing a single template is shown in Listing 4.4.

Listing 4.4: Output for the full list of prompt templates with attributes

```
[
  {
    "politeness_level": "neutral",
    "sentence_depth": "shallow",
    "sentence_type": "declarative",
    "template_type": "pos",
    "text": "My first {A} template test."
  }
]
```

To simplify template selection based on the properties the RL agent determines, another app route allows filtering for a prompt template based on a combination of attributes. An exemplary call for the route could look like this:

```
http://localhost:5556/api/template?sentence_type=1&politeness_level=2&sentence_depth
    =1&template_type=1
```

As shown in Listing 4.5, the response returns the list of fitting text candidates. In the context of this work, when a full filter combination is given, exactly one template fulfilling all criteria is always returned in the response.

Listing 4.5: Output for prompt template text

```
[
  "My first {A} template test."
]
```

A utility route is used to provide the available action space, consisting of the template selection properties the agent can utilize, in a compact way. The selection options, fully shown in Listing 4.6, display the option identifiers along with the representational label.

Listing 4.6: Output for prompt template text

```
{
  "politeness_levels": {
    "1": "polite",
    "2": "neutral",
    "3": "impolite"
  },
  "sentence_depths": {
    "1": "shallow",
    "2": "medium",
    "3": "deep"
  },
  "sentence_types": {
    "1": "declarative",
    "2": "interrogative",
    "3": "imperative"
  },
  "template_types": {
    "1": "pos",
    "2": "wildcard"
  }
}
```

Lastly, for the topic groups, another handy route is implemented to allow accessing all groups along with their topic entries. The JSON objects, as exemplarily shown for one topic group in Listing 4.7, can be utilized during the RL process to model the input values for the first sub-process and the topic inputs for mask templates easily.

Listing 4.7: Output for the topic group collection, example for a single group with its entries

```
{
    "entries": [
        "computer",
        "internet",
        "software",
        "hardware",
        "AI"
    ],
    "id": 1,
    "label": "Technology"
}
```

4.3 Readability Evaluation Unit Service

The implementation of the readability evaluation unit is straightforward, following the methodology given in Section 3.4.3. Readability metrics are manually implementing, allowing the system to gain maximum flexibility in order to realize the specified normalization procedure later.

The service is once again implemented as a Flask microservice, allowing a flexible usage of the interface to evaluate texts generated in LLM context. The text that is meant to be assessed is passed to the service using the POST body of the HTTP request. The following command input shows an example of a readability request submitted from the local system where the readability service has been set up.

```
curl -H 'Content-Type: application/json' \
    -d '{ "text": "This is a test message. It is used to validate the readability
        evaluation unit." }' \
    -X POST \
    http://localhost:5557/api/readability
```

The service provides output for all separate readability metrics taken into account for this work. The calculations are divided into the raw readability values, representing the direct output numbers of the individual metrics, as well as normalized values representation by Equation 3.4 and Equation 3.5. Parameters for n and d are set as given in Table 3.3. The calculation also contains the mean value of all seven normalized readability scores, which will be utilized as the main source of reward signal for the RL agent. Listing 4.8 shows how an exemplary output looks like for a text that is made up of only two sentences.

Listing 4.8: Output of an exemplary readability evaluation that consists of only two sentences


```
{
    "mean_normalized": 0.26084986827103007,
    "normalized": {
        "ari": 0.9762406015037594,
        "coleman_liau": 0.21662337662337644,
        "dale_chall": 0.6807110389610389,
        "flesch_kincaid": -0.3443736263736259,
        "gunning_fog": 0.19591836734693885,
        "lix": 0.0272108843537414,
        "smog": 0.07361843548198141
    },
    "raw_values": {
        "ari": 4.274285714285714,
        "coleman_liau": 7.69142857142857,
        "dale_chall": 8.495128571428571,
        "flesch_kincaid": 42.61571428571432,
        "gunning_fog": 11.371428571428572,
        "lix": 35.57142857142857,
        "smog": 10.125756701596842
    },
    "text": "This is a test message. It is used to validate the readability
        evaluation unit."
}
```

4.4 Reinforcement Learning Services

The last major service components deal with the RL algorithms used for the implementation. Therefore, two separated components play an important role. The *training documentation service* is crucial for serializing all steps conducted during training. This way, decisions favored by the RL algorithm will be analyzed in the evaluation described in Chapter 5. Additionally, model snapshots can be regularly updated within this service as well. Core procedures of the RL are realized within the *learning agent service*, providing the training procedure for the selection of a prompt mask and prompt template insertion based on the topic input group. Lastly, the *prompt inference service* offers a primitive means to test fully-fledged models in order to obtain full prompts that can be directly passed to LLMs.

Implementation details for all the named components will be detailed in the following.

4.4.1 Training Documentation Service

Training documentation aims to record every step of training progress conducted by the RL agent. Therefore, all data related to a training run are saved. Starting with general information for a *training run*, the system accepts all information regarding system setup and hyperparameters used for the entirety of the run. Individual training steps store various data for each sequence that selects a new prompt template and fills all mask spaces to provide an input prompt for LLM text generation. Data saved are, besides the main data model, spread into separate tables with specific purpose:

- *Template selection properties*: Attributes related to the first part of the RL process responsible for selecting the prompt template, depending on the input group that has been selected as input.
- *Mask fills*: Information for each selection process conducted during the iterative mask filling for the selected prompt template.
- *Reward properties*: All readability information along the normalized mean score used to deduce the reward signal for the RL agent.

The database models used to represent the described structures are visualized in the diagram shown in Figure 4.5. Properties that are related to the word pool (see Section 4.2 for reference) are referenced using their unique identifiers, thus limiting data redundancy.

The documentation is done by using two POST interfaces. The first interface allows creating an entirely new training run. This endpoint, accepting the various hyperparameters for this run in the request body, returns the database identifier for the newly created training run. During training, new training steps (in other words, individual training episodes) are inserted using the second interface. To link the episode with the current run, the already generated training run ID is passed using the request body. Additionally, training step data include attributes the agent selected for the prompt template, all information regarding filled in words, normalized and raw reward properties as well as general data such as the current episode, the selected topic and both the used prompt and the LLM-generated text.

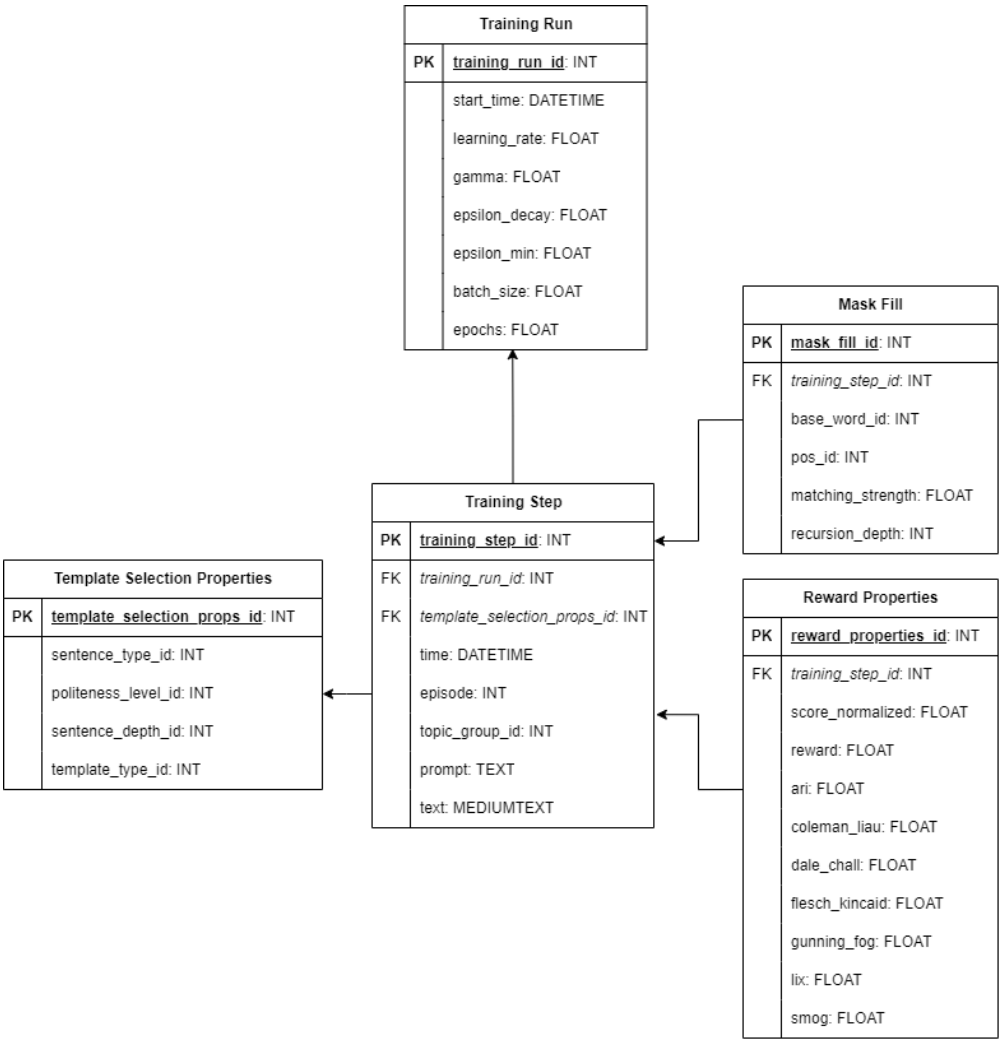

Figure 4.5: Entity-relationship diagram for the word pool database setup

4.4.2 Learning Agent Service

This service, responsible for selecting both prompt template and words to fill all mask spaces, is the major instrument to carry out the RL training. Therefore, the services described earlier in this chapter are utilized to provide the set of actions the agent can select from and at the same time create the environment modeled with the preexisting topic groups. This section summarizes implementation details and the training progress, which is used in Chapter 5 to evaluate this work.

Large Language Model Interaction

For the implementation rework, Llama 3.1[15] has been chosen as the language model. Meta does not only publish various models as open source software, but even smaller versions of the model showed promising results on various state-of-the-art benchmarks [60]. Due to the limited time and resources of this work, the smallest model version which has 8 billion parameters has been chosen as the inference LLM for training.

[15] https://ai.meta.com/blog/meta-llama-3-1/

For distribution, Ollama[16] has been used. This practical tool can directly fetch the 8b parameter version of Llama 3.1 and allows interacting with the model using command line input, a web overlay or a simple interface that can be accessed using HTTP requests. As the LLM is utilized as another type of service used by the RL-based client, the latter is employed for inference. As already outlined in previous sections, fleshed out prompts are sent to the model using HTTP POST requests, allowing the client to access generated responses directly in order to pass them to the readability evaluation service.

Reinforcement Learning Agent Implementation

To enable the ML frameworks to directly make use of the word pool and template data, two distinct RL agents are employed: One for template selection (template agent) and another for filling in words from the word pool (mask agent). Each agent has a separate action space and corresponding neural network architecture designed to manage its specific task efficiently.

The template agent is responsible for selecting the prompt template. Its action space is a set of all possible combinations of template attributes, such as sentence type, politeness level, sentence depth, and template type. The template agent operates on a simplified input state that may include just the topic group ID or template ID, making the agent's input space one-dimensional.

The mask agent fills the masks within the selected template with appropriate words from the word pool. Its action space includes all words that can be selected to fill in the template, constrained by the mask type (e.g., part-of-speech constraints or wildcard). The mask agent uses a more complex state representation that includes multiple features of the words already selected and other template-specific attributes.

The mask agent utilizes the word pool, which serializes the following attributes:

- corpus ID and one-hot encoding for whether the corpus is a question corpus or not
- POS ID
- matching strength, scraping recursion depth index, and parent ID (only if the word is a synonym)

Besides the named word pool data, available prompt template properties are accessible using their labels and the respective amount of possible actions the agent can select from. This way, a numerical identifier for each attribute can be picked during a training run in order to obtain a prompt template that is later used for LLM inference without further preprocessing steps. To enhance the training stability, all named input features are normalized before being passed into the DQN model. Normalization ensures that each feature contributes equally to the agent's decisions, preventing bias by features with larger numerical ranges.

To start a training run, all the features needed are initialized using a *training configuration*. The configuration, saved as a YAML file within the distributing monolithic *learning environment controller*, holds various hyperparameters as well as URIs needed to access services and the LLM interface. Related data are loaded before the training run starts, enabling the agent to easily access variables

[16] https://ollama.com/

when needed. Another parameter defines a *prompt suffix* that is always attached to the end of a generated template. This is done to ensure that all prompts follow a certain structure and have a minimum length, independent of the preceding prompt template the agent has filled.

The configuration setup used for the model training of this work is shown in Listing 4.9. This configuration at the same time shows the used hyperparameter values.

Listing 4.9: Training configuration for the core training run of this work

```
learning_rate: 0.001
batch_size: 64
epochs: 10000
gamma: 0.99
epsilon_min: 0.01
epsilon_decay: 0.997
target_network_update_frequency: 10
word_pool_service_url: "http://localhost:5556/api"
readability_service_url: "http://localhost:5557/api"
llm_url: "http://141.55.228.245:11434/api"
llm_model: "llama3:8b"
prompt_suffix: "Start with a general description and end with specific details.
    Write at least 200 words of only plain text."
```

The template agent and mask agent are implemented using the Keras library, each with a distinct neural network architecture tailored to their specific tasks.

The template agent consists of a simpler neural network with two dense layers. The first dense layer has 24 neurons and utilizes a ReLU activation function. This is followed by another dense layer with 24 neurons and a ReLU activation function, and finally, an output layer with a linear activation function that outputs Q-values for each possible action related to one property of a possible prompt template attribute.

The mask agent utilizes a more complex neural network due to its more detailed input state. The network starts with a dense layer of 64 neurons with a Leaky ReLU activation function (alpha=0.01), followed by a batch normalization layer. This is succeeded by several dense layers with increasing neurons (128 neurons) and additional dropout layers to prevent overfitting. The final output layer uses a linear activation function to output Q-values corresponding to the action space of selecting words.

A new training episode starts with selecting a topic group randomly. The group index serves as the input to the template agent, defining its state space. The template agent then selects a prompt template from all possible combinations of sentence type, politeness level, sentence depth, and template type. The selected template is used as the base for the next phase. Next, the mask agent iteratively selects words to fill in all masks within the selected template. The state space for the mask agent includes the topic group ID and the attributes of previously selected words. Depending on the mask type (thus, POS constraints or wildcard), the action space is appropriately limited to ensure relevant word selection.

Using the chosen template as a base, the next phase of RL iteratively selects words for all masks within the template. Input layer parameters for this agent are the topic group ID passed from the initial input as well as the topic group ID. Moreover, six neurons of the input layer are reserved per already inserted word. Each neuron represents one property of a base word. To allow a maximum of five consecutive selections, 24 neurons are reserved in total for up to four words selected beforehand. Thus, when the first word is filled in, none of the reversed neurons is used while on the second mask space the six properties of the first selected word are passed to the input layer, and so on.

The training process integrates feedback from a readability evaluation service, which provides normalized readability scores for text generated by a LLM. These scores, which have a value range between -1 and 1, are multiplied with a hyperparameter number and squared while keeping their sign in order to reinforce good rewards and punish bad scores even more rigorously. For training, the value range is therefore scaled up from -9 to 9.

The workflow of the training process, just like with the preliminary approach, incorporates various implementation details. The agents employ an epsilon-greedy strategy, where the epsilon value decays over time, gradually shifting the focus from exploration (random actions) to exploitation (greedy actions based on the learned policies).

A separate target network is used to update the Q-values regularly. As explained in Section 3.4.4, this technique primarily aims to improve training stability. The target network shares the same architecture as the corresponding main model. The separation of the target network from the root model ensures that updates can be controlled more easily. The target network is updated periodically by copying the weights from the main model. This update occurs after a set amount of episodes, as specified by the training configuration parameter. By updating the target network at regular intervals rather than after every training step, the implementation helps to smooth the training process and prevent the propagation of errors that might arise from the main model's frequent updates. Experiences (state, action, reward, next state) are stored in a replay buffer to stabilize training by avoiding correlation between consecutive experiences.

Periodic updates to the target network and model checkpoints are saved during training to retain the best-performing models. Instead of saving DQN weights in separate HDF5 files, the new implementation saves the entire state of both agents using the pickle module[17]. This method merges the weights and other relevant parameters into a single pkl-file. These checkpoints enable resuming training from a specific episode. Additionally, the best-performing models are identified based on the rewards obtained and saved together in a single file. These models can later be used for testing purposes by the prompt inference service (see Section 4.4.3 for details).

For training, Llama 3.1-8b was distributed on a separate system using the official Ollama Docker image[18]. This enables the learning agent service to conduct all ML-related tasks locally while outsourcing the part of time-consuming text generation entirely to another machine. Due to the level of complexity of prompt generation and filling using the word pool, over generating the text to be

[17] https://docs.python.org/3/library/pickle.html
[18] https://hub.docker.com/r/ollama/ollama

evaluated up to RL agent training, the workflow was more time-consuming than with the preliminary approach. Training time for 10,000 episodes was about 85 hours, including logging all data in the training documentation database and saving model snapshots.

4.4.3 Prompt Interference Service

Prompt inference directly leverages top-performing model files saved during training in order to provide an interface that can be used to generate new prompts that grant better text readability when used in LLM context. By loading the saved weights of both agents, the service constructs prompts based on the topic group input. The agents predict the most suitable template and words to use by decoding the features returned by both individual agents, thus representing a single training episode where both prompt template and mask filling words are selected.

Once again, the service has been implemented using Flask. A single POST interface allows the transmission of a topic group ID used for the prompt selection, as well as the topic string that can be user-defined in order to allow a flexible use of the superior topic group. The selected topic is automatically set into the topic space within the selected template. The prompt that is returned by the response can consequently be used for LLM interaction in order to generate texts with an optimized readability.

4.5 Exemplary Distribution of the Systems using a Web Application

To exemplarily showcase how all components developed for this work could be used for a practical scenario, a testwise distribution is rolled out. A simple web application is used in order to interact with the inference service and an LLM to generate topic-specific, easy-to-read texts using the results of this work.

For implementation, React[19] is used. The application consists of a simple page which allows selection for a superior topic group using a dropdown menu. Entries for the menu are dynamically loaded using the topic group endpoint within the word pool interface. The specific topic can be directly entered using a text field input. Both input elements are shown in Figure 4.6.

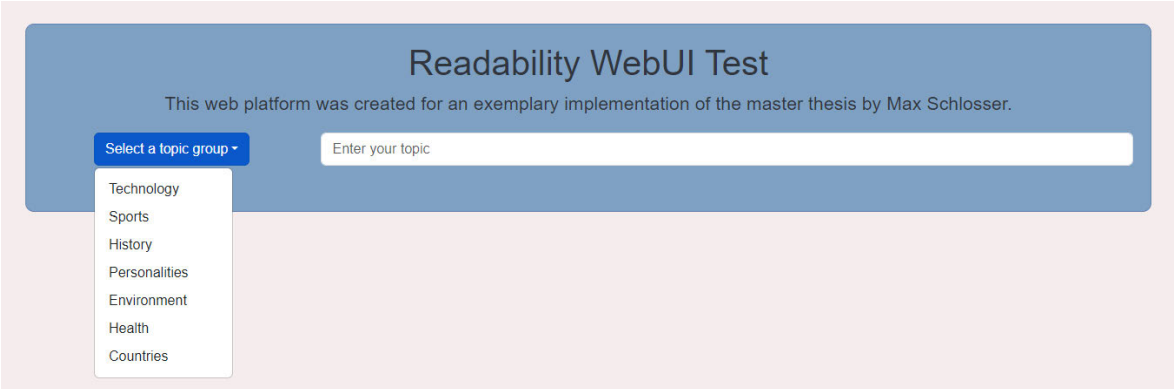

Figure 4.6: Input elements for the web application

[19] https://react.dev/

When both elements are set, text generation can be started using another button. When doing so, the ID of the selected topic group as well as the input term are sent to the prompt inference service, returning a full prompt in response. This prompt is passed to the LLM, in this case again Llama 3.1:8b for test purposes. The generated text is assessed in terms of readability to give an impression of how the automatically generated prompt influences the used metrics. As shown in Figure 4.7, raw and normalized metrics as well as the full generated text are rendered on the web page.

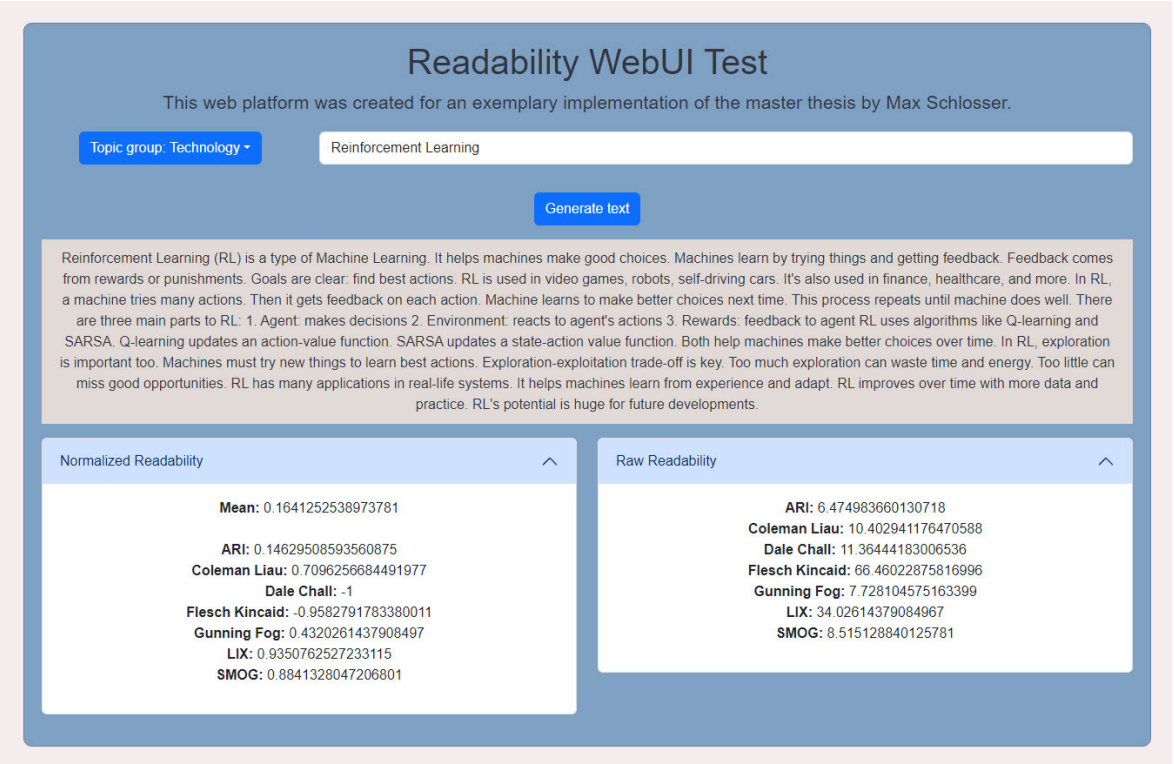

Figure 4.7: Text generation and readability evaluation in the web application

5 Evaluation

In this chapter, results of the training process are evaluated in order to deduce statements about the effectiveness of the implementation rework. To do so, hypotheses formulated in Section 3.5 addressing the different factors in relation to the readability recorded during training will be evaluated. By using statistical tests, the properties utilized in this work as well as RL in general are assessed in terms of potency on prompt composition. Attributes that show significant effects are separately analyzed in order to find out which selections show most promising results for text generation.

5.1 Data Analysis and Testing Initial Hypotheses

All training data saved in the training documentation service (see Section 4.4.1 for details) have been included for testing the hypotheses. The following approaches were chosen to evaluate the hypotheses:

1. H_2**:** Compare readability scores of generated texts from early and late training episodes. Define two groups, E_1 (early episodes) and E_2 (late episodes), and compare their normalized readability scores using the readability metrics applied in this work.
2. H_3**:** Group episodes by matching strength values and analyze mean readability scores across these groups to identify correlations.
3. H_4**:** Compare readability scores across different recursion depth levels to determine the impact on text readability.
4. H_5**:** Categorize episodes based on the dominant part of speech used and compare readability scores for different categories.
5. H_6**:** Group episodes by types of morphemes used and evaluate the differences in readability scores.
6. H_7**:** Compare readability scores across templates with varying sentence depths to assess their impact.
7. H_8**:** Group episodes by politeness level used in templates and compare readability scores across these groups.
8. H_9**:** Analyze readability scores based on different sentence types used in templates to determine their effects.
9. H_{10}**:** Compare readability scores for episodes using POS-specific mask spaces versus those using wildcard mask spaces to evaluate effectiveness differences.

To evaluate Hypothesis H_2, the dataset was divided into early and late training episodes based on episode numbers, with early episodes E_1 being the first 2500 and late episodes E_2 being those from episode 7500 onwards. This segmentation captures the very early and late stages of training. A t-test was performed to compare the readability scores between these two groups, assessing whether there was a significant improvement in readability over time. For Hypothesis H_3, the dataset was grouped based on the average matching strength of words within each episode, and an ANOVA test was used to detect significant differences in readability scores across these groups. Hypotheses H_4 through H_6 involved grouping the dataset by the average recursion depth, the most common POS, and the most common morpheme type per episode, respectively. ANOVA tests were then conducted

to examine whether there were significant differences in readability scores across these categories. Similarly, Hypotheses H_7 through H_9 used ANOVA to analyze the effects of template attributes such as sentence depth, politeness level, and sentence type on readability scores. Finally, Hypothesis H_{10} compared readability scores between templates using POS-specific masks and wildcard masks by performing a t-test to determine if there were significant differences between these two types of templates.

All tests are conducted with a significance level of 5% (0.05). The following results were computed from the statistical analyses performed to evaluate the hypotheses regarding the impact of RL and word selection on text readability:

1. H_2**: p-value:** 1.071e-281
   Since the p-value is extremely less than the significance level of 0.05, the null hypothesis is rejected. This indicates that there is a strong statistically significant improvement in text readability over time due to RL.
2. H_3**: p-value:** 0.011
   The p-value is slightly below the significance level of 0.05, leading to reject the null hypothesis. This suggests a statistically significant relationship between higher matching strength and improved readability.
3. H_4**: p-value:** 0.201
   The p-value is greater than the significance level of 0.05, so the null hypothesis cannot be rejected. This indicates that there is no statistically significant effect of recursion depth on text readability.
4. H_5**: p-value:** 0.045
   As the p-value is below the significance level of 0.05, the null hypothesis is rejected. This suggests that the part of speech of selected words has a statistically significant impact on readability.
5. H_6**: p-value:** 0.081
   Since the p-value is greater than 0.05, the null hypothesis cannot be rejected. This implies that there is no statistically significant effect of morpheme types on readability.
6. H_7**: p-value:** 0.019
   The p-value is less than 0.05, leading to reject the null hypothesis. This indicates a statistically significant impact of sentence depth in templates on readability.
7. H_8**: p-value:** 0.029
   Given that the p-value is below 0.05, the null hypothesis is rejected. This suggests a statistically significant effect of politeness level in templates on readability.
8. H_9**: p-value:** 0.030
   The p-value is less than the significance level of 0.05, so the null hypothesis is rejected. This indicates a statistically significant influence of sentence type in templates on readability.
9. H_{10}**: p-value:** 0.103
   The p-value is greater than 0.05, meaning the null hypothesis cannot be rejected. This suggests there is no statistically significant difference between the results of POS-specific masks and wildcard masks.

5.2 Special Consideration of Significant Variables

For the selection properties that showed significant differences in terms of the generated reward, special attention is paid to the individual factors that the RL agent learned in order to stabilize or improve readability.

5.2.1 General Influence on Readability

H_2 showed that there is a significant difference between normalized reward distribution when comparing the first 2500 and last 2500 episodes of training. Figure 5.1 shows the normalized readability scores per episode during training, with an orange overlay representing mean scores for every ten episodes.

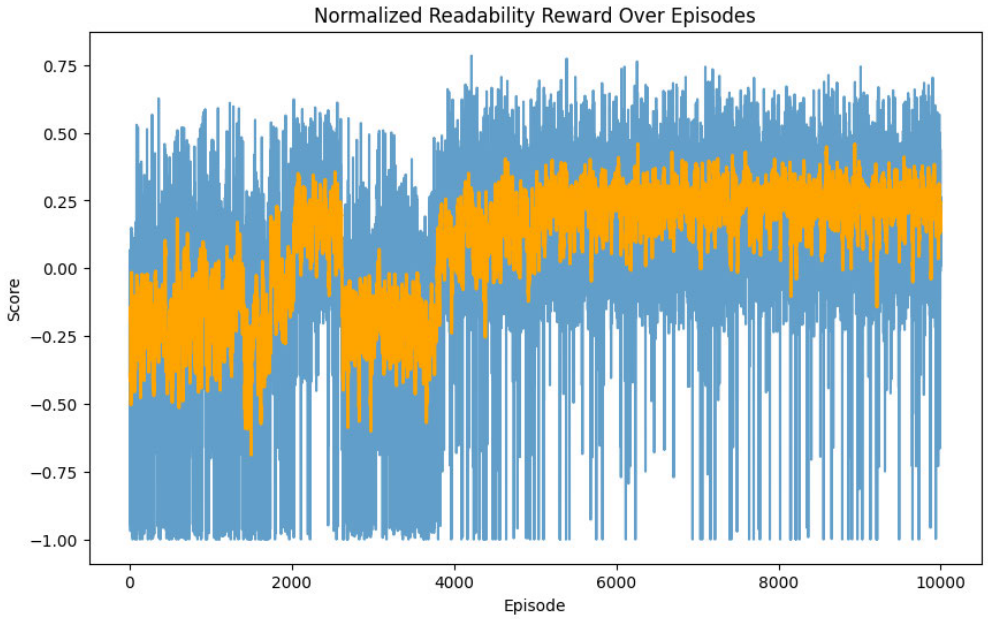

Figure 5.1: Score distribution over a training run of 10,000 episodes: All data points (blue) and the baseline composed of 10 consecutive score means (orange)

Important statistical data for normalized readability scores of both groups are summarized in Table 5.1.

Table 5.1: Summary statistics for the normalized readability score in the first and last 2500 episodes

	First 2500 Episodes	Last 2500 Episodes
Mean	-0.115	0.236
25%	-0.399	0.108
Median	-0.023	0.302
75%	0.111	0.396
St. Deviation	0.381064	0.256
Min	-1	-1
Max	0.626	0.744

Figure 5.2 breaks down how each of the readability metrics changed over the training curse. The graphs are created by aggregating mean values of 200 consecutive episodes to significantly improve recognizability. As visible in the graphs, some metrics are relatively steady during training while some readability indices could be influenced heavily using RL.

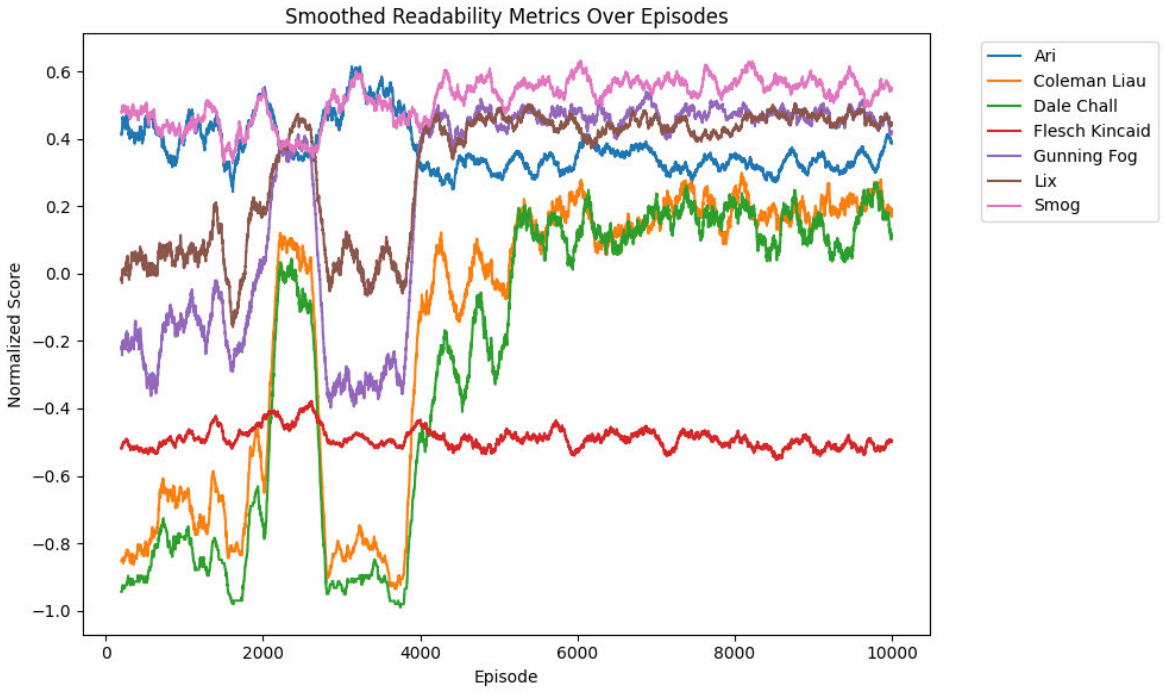

Figure 5.2: Normalized readability metric distribution over a training run of 10,000 episodes, smoothed for every 200 episodes

5.2.2 Influence of Matching Strength

The acceptance of H_3 suggests that the matching strength determined during recursive synonym scraping affects the normalized readability score. Thus, each matching strength category is set into relation with the mean normalized score obtained when choosing a base word with the given type, as well as the amount of word choices having this type. Table 5.2 summarizes the statistics.

Table 5.2: Mean normalized reward and agent selection count by word matching strength

Matching Strength	Mean Normalized Reward	Count
0.0 (weakest)	0.345002	15972
0.5 (medium)	0.029424	2196
1.0 (strongest)	-0.139649	8694

5.2.3 Influence of Part of Speech Usage

With the acceptance of H_5 a correlation between a base word's POS and the resulting normalized reward can be deduced. Table 5.3 links the mean normalized score and the amount of agent selections per POS across all episodes.

Table 5.3: Mean normalized reward and agent selection count by word part of speech

Part of Speech	Mean Normalized Reward	Count
Adjective	-0.520227	1877
Noun	0.229400	20747
Verb	0.136380	4238

5.2.4 Influence of Sentence Deph

A significant influence of the sentence depth level on the normalized readability score is accounted by the acceptance of H_7. Table 5.4 once again shows how often each level was chosen for prompt templates and what the mean resulting reward looked like.

Table 5.4: Mean normalized reward and agent selection count by sentence depth

Sentence Depth	Mean Normalized Reward	Count
Deep	-0.423357	586
Medium	0.176313	1552
Shallow	0.214510	7862

5.2.5 Influence of Politeness Level

H_8 is confirmed, thus a significant correlation between a prompt template's politeness level on the resulting readability score is given as well. The mean reward obtained per politeness level as well as the amount of template choices using this politeness level are given in 5.5.

Table 5.5: Mean normalized reward and agent selection count by politeness level

Politeness Level	Mean Normalized Reward	Count
Impolite	0.206384	7662
Neutral	-0.004835	693
Polite	0.088408	1645

5.2.6 Influence of Sentence Type

Lastly, due to the acceptance of H_9, the connection between the sentence type of the used template and the resulting reward is assessed. Table 5.6

Table 5.6: Mean normalized reward and agent selection count by template sentence type

Sentence Type	Mean Normalized Reward	Count
Declarative	-0.061526	1109
Interrogative	0.144293	7693
Imperative	-0.068028	1198

6 Discussion

Results obtained during training as well as their evaluation conducted in Chapter 5 allow interpretation. Thus, this chapter discusses the grade at which RL algorithms utilized in this work were capable to semi-automatically construct prompts in order to maximize readability metrics for LLM-generated texts. Both general implications and specific properties statistically relevant are included. The last section highlights limitations of the implemented methodology. Finally, approaches that could build on this work for future research are suggested.

6.1 General Implications

The results support the hypothesis that RL can improve text readability over training time. In particular, Hypothesis H_2 demonstrated a statistically significant improvement in readability scores between early and late training episodes. This suggests that RL algorithms effectively learn to optimize prompt composition by filling pre-defined spaces in a template in a way that enhances readability. The use of a target network appears to have stabilized training in the reworked approach, ensuring that improvements were consistent across training episodes. The significant improvement in normalized readability score, as indicated by the extremely low p-value, highlights the robust nature of RL in refining language models to generate more readable text over training periods. The results however, just like the preliminary experiment, indicate that incorporation of RL leads to a stabilization of readability of generated text more than a full-fledged improvement. This is illustrated by the lower standard deviation and higher mean reward in later training episodes, while no considerable improvement in terms of maximum reward can be observed. While later episodes still had some samples generating the lowest possible normalized score of -1, maximum scores peaked at about 0.75 while the possible value range reaches up to 1.

When looking at the distribution of each metric over the training run, it becomes obvious that some metrics are heavily influenced by the RL approach, while others seemed almost completely stable. The metrics that were influenced most are Coleman Liau, Dale Chall, Gunning Fog and LIX. On the other hand, ARI, Flesch Kincaid and SMOG did not improve noticeably or even decreased over the training run. The differing impacts can be attributed to the statistical components involved in each metric's formula. The metrics that were most influenced all heavily rely on word length, sentence length, or the frequency of complex words. These factors are directly affected by the RL-driven word selection and sentence structuring processes, which aim to optimize readability by adjusting these variables. In contrast, metrics like ARI, Flesch-Kincaid, and SMOG are more focused on syllable counts and polysyllabic words. These aspects are less sensitive to changes in prompt composition and are not as directly targeted by the RL strategy, leading to their relative stability or even decline over the training run. Thus, the model appears to be more effective at optimizing metrics that prioritize simple word and sentence structures over those that emphasize syllable-based complexity.

These findings have important implications for the usage of (semi-)automated text generation. As language models continue to be integrated into various applications, the ability to automatically optimize prompts through RL could lead to more effective and user-friendly interactions. The fact that

readability can be systematically enhanced through such methods suggests that similar techniques could be applied to other aspects of language model output, such as relevance, conciseness, or specific tasks.

In addition to the general results, the chosen implementation turned out to be extremely practical when bringing together all developed components in order to efficiently conduct and monitor all necessary preprocessing steps and the training sequence. Using a distributed approach with a monolithic core component allowed to keep control over the entirety of processes included in the workflow, while at the same time allowing for efficient maintainability or even replaceability of single services. Scalability was not the primary focus of this work, as the main goal was to establish a solid understanding of how different architectural components and RL strategies could influence text readability, without the additional complexity of scaling considerations. However, the chosen architecture naturally supports scalability, which allows for individual components to be scaled independently based on workload requirements.

6.2 Implications of Specific Properties used for Training

The evaluation of specific properties provided further insights into how different attributes for words or prompts accounted in this work impact text readability. Hypothesis H_3 showed that the selection of words based on matching strength significantly affects readability. Words with a very low matching strength (therefore words that are used less commonly as synonyms) showed the greatest reward improvements, followed by the words with moderate usualness. Although being used quite often, words with a strong matching strength showed the lowest mean reward. Their high usage can be explained with random selection moves during a phase where epsilon was high, as this group was overrepresented due to the base corpora accounting base words with a matching strength of 1.

Moreover, Hypothesis H_5 confirmed that the POS of selected words impacts readability, with nouns and verbs contributing more positively than adjectives. This result aligns with linguistic theories that emphasize the importance of content words (nouns and verbs) over function words in maintaining clarity and reducing cognitive load for readers. Here, nouns, followed by verbs, resulted in the greatest improvements for rewards. Nouns were also the most chosen factor by the agent, which may also be due to the large number of nouns in the word pool.

Additionally, the influence of sentence depth (H_7) and politeness level (H_8) in templates was found to be significant. Shallow and medium sentence depths were more effective in improving readability, suggesting that simpler sentence structures are generally easier to process, while complicated input sentences negatively influence readability results. The politeness level also had an unexpected effect: Impolite templates were associated with higher readability scores, which could be due to the directness and clarity often found in less polite language reciprocated by the LLM answer. This also supports the work of Yin *et al.*, stating that some LLMs might respond with shorter sentence structures, which explains better readability scores [25].

Hypothesis H_9, concerning sentence type, showed significant difference in readability outcomes. Here, the group of interrogative sentences showed highly positive results when compared with the other two groups. Declarative sentences and imperative sentences were used similarly and also

led to almost the same rewards in relation. Thus, a connection between imperative statements and shorter outcomes cannot be confirmed, which is contrary to the high positive influence of impolite language.

Hypotheses H_4, H_6 and H_{10} concerning recursion depth in terms of scraping, types of morphemes and the type of mask spaces in a template (POS-driven or wildcard) were rejected. This suggests that, at least within the scope of this study, these factors are less critical than others and that more attention should be given to optimizing word choice and sentence structure rather than these attributes.

6.3 Limitations and Future Directions

Despite the promising results, several limitations must be acknowledged. Firstly, the study primarily focused on readability as the sole measure of text quality. Readability, while important, does not encompass all aspects of effective communication, such as informativeness, engagement, or emotional impact. Future research could benefit from incorporating a more comprehensive set of evaluation metrics to provide a fuller picture of text quality. This could also involve user-driven assessment of texts in order to gain another layer of insight of how readability is perceived by humans instead of readability metrics. Building on this idea, RL could be influenced through human feedback using techniques of RLHF. Related to this, the existing implementation could be leveraged to explore scalability aspects in greater depth, using the modular nature of different services to handle larger datasets, increase computational resources dynamically, or expand the system's capabilities to process more complex tasks.

It is also conceivable that the general approach suggested in this work can be transferred to other task-specific scenarios. For example, another objective score calculation representing quality for a designated task could replace the readability unit to provide a reward signal. Thus, the agent could be trained to fill in other words in new prompt templates in order to optimize for a particular task. This idea aligns with the research of Deng *et al.*[42]. However, when comparing the chosen implementation approach, it becomes clear that the improvements achieved are rather small in relation to the simple means in the preliminaries given in Section 3.3. Thus, it is difficult to make statements about the general transferability of the chosen methodology to other domains.

Furthermore, the RL model's reliance on specific readability metrics, which were normalized based on predefined parameters, introduces a level of bias. Other contexts may require alternative metrics or adjustments to the normalization process, or could even include other extensive metrics which include additional criteria for measuring readability. While the use of a RL-based approach has shown substantial benefits, the complexity and computational cost associated with training such models cannot be overlooked. Further research into more efficient RL algorithms, possibly incorporating techniques such as hierarchical RL, curriculum learning, or actor-critic methods, could help in reducing these costs and making the approach more accessible and even better-performing for other applications.

Moreover, the methodology chosen to construct the word pool along with the selected criteria used to annotate words and templates might be error-prone and potentially limiting in terms of other options. Future research could investigate how altering processes constructing the word pool or modified

templates would result in changes in the model's performance. Beyond that, the impact of linguistic properties such as matching strength, POS, and sentence depth warrants deeper exploration. Understanding why certain properties enhance readability could provide insights into the processes involved in text comprehension and train more sophisticated models of text generation that better mimic human language use.

7 Conclusion

In this thesis, the optimization of text readability using RL within the context of semi-automatic prompt engineering for LLMs was explored. The primary focus was on enhancing readability by manipulating prompts through the use of various linguistic properties and RL techniques. This approach combined advanced ML algorithms with traditional linguistic analysis to develop a novel method for automatically filling pre-defined, criteria-based prompt templates with words from a methodically constructed pool of terms.

The implementation of RL in text generation was based on the principle of optimizing prompts to achieve better readability scores, as measured by several established readability indices. The study demonstrated that it is possible to significantly improve the consistency of text outputs from LLMs by carefully selecting words and constructing prompts based on readability metrics. This was particularly evident when comparing the RL-optimized prompts against randomly generated prompts or results from early training episodes, where the optimized approach resulted in a reduction in the standard deviation of readability scores, highlighting an increase in text readability coherence. However, at the same time, the results show that the process tends to lead to a stabilization of the output readability, but the greatest possible scores are not achieved automatically.

Several experiments were conducted to analyze the impact of different linguistic properties on readability. The results indicated that properties such as sentence depth, politeness level, and sentence type have significant effects on readability outcomes when accounted in the corresponding prompt. For instance, templates with shallow sentence depth and impolite language were associated with higher readability scores. These findings suggest that prompts using simpler sentence structures and direct language contribute to more readable texts when being used with LLMs. Additionally, the study found no significant impact of certain properties, such as scraping recursion depth in word selection and morpheme types, on readability, suggesting that these factors might be less critical for optimizing text generation in the context of LLM text generation.

The technical implementation leveraged a distributed architecture with a modulithic core to enhance scalability and maintainability, employing microservices for various components like word pool analysis and distribution or readability evaluation. This architectural choice opened up asynchronous training and evaluation processes, significantly improving the efficiency of the system.

Despite these advancements, the study also pointed out some limitations. One major limitation was the focus on readability as the sole measure of text quality. While readability is crucial, it does not capture all aspects of effective communication, such as engagement or emotional resonance. Future research could expand on this work by incorporating additional metrics to evaluate these other dimensions of text quality. Additionally, the RL model's dependency on specific readability metrics introduces a bias that might not be universally applicable across different contexts. Future studies could explore alternative metrics or develop more generalized models that account for a wider range of textual attributes.

Looking forward, there are numerous potential approaches for extending this research. One possible direction is to refine the RL-based optimization process by incorporating more sophisticated reward functions that are targeted at specific text generation tasks. Furthermore, exploring other ML techniques could provide additional improvements in training efficiency and model performance.

Appendix A: Experiment: Word Pool (selected manually)

a, abnormal, about, absolute, abundant, accurate, acquired, adamant, adjective, advanced, adverb, agreed, ambiguous, ample, an, and, angle, apostrophe, apparent, applicable, appointed, approach, appropriate, approximate, arduous, are, article, as, aspect, at, attribute, atypical, be, begin, beginner, believable, block, bountiful, broad, but, by, can, certain, challenging, chaotic, characteristic, chosen, citation, clarity, clause, clear, coherence, coherent, collective, colon, colossal, comma, common, communication, complex, complicated, compound, comprehension, compulsory, concept, concise, concluded, confident, conjunction, connection, considerable, consistent, constant, continuous, convincing, copious, correct, create, credible, critical, crucial, dash, decided, deep, defined, definite, demanding, demonstration, design, designated, destination, detail, detailed, determined, difficult, discontinuous, disordered, diversity, do, doubtful, each, easier, easy, elaborate, elected, element, ellipsis, elucidation, enhance, enormous, essential, established, evident, exact, exacting, example, exclamation, exhaustive, explanation, explicit, exposition, expression, extensive, external, extreme, extrinsic, factor, feature, firm, fitting, fixed, fluctuating, for, format, framework, from, full, fundamental, general, generate, generous, gigantic, grammar, had, hard, harsh, have, he, her, hidden, his, hot, how, huge, hyphen, I, idea, if, illogical, illustration, immense, immovable, impersonal, implausible, implicit, important, improve, in, inaccurate, incoherent, incomprehensible, inconsistent, incredible, indefinite, indispensable, inexorable, inflexible, inherent, insight, instance, internal, interpretation, intricate, intrinsic, irrational, irregular, irrelevant, is, it, laborious, language, large, lavish, layout, less, liberal, like, location, logical, long, main, major, make, mammoth, mandatory, manner, many, mark, method, minor, model, moderate, more, necessary, non-standard, nonsensical, noob, normal, noun, objective, obligatory, obscure, of, on, one, onerous, opted, optional, or, ordered, organization, organized, origin, paragraph, particular, pattern, perception, period, persistent, personal, perspective, pertinent, phrase, picked, plan, plausible, plentiful, position, precise, predictable, preposition, primary, principal, profuse, pronoun, proper, property, prototype, punctuation, quality, question, quotation, random, rational, redundant, reference, regular, relative, relentless, relevant, reliable, required, resolute, resolved, rich, right, rigorous, secondary, see, selected, selection, semicolon, sensible, sentence, set, settled, severe, shallow, she, shorten, significant, simple, simplify, so, some, sophisticated, source, specific, specified, stable, steady, straightforward, strategy, strenuous, strict, stringent, structure, structured, style, subjective, substantial, suitable, superficial, syntax, system, systematic, tactic, technique, template, tertiary, text, that, the, their, them, then, there, these, they, thing, this, thorough, time, tiny, to, tough, trait, transition, tremendous, trivial, typical, unbelievable, uncertain, uncommon, uncompromising, unconvincing, understand, understandable, understanding, unimportant, universal, unnecessary, unpredictable, unreliable, unstable, unstructured, unusual, unwavering, unyielding, up, use, useful, usual, vague, variable, variation, vast, verb, view, vital, vocabulary, was, way, we, what, which, wide, will, with, word, would, write, you

Appendix B: Experiment: Prompt Template Combinations

In each prompt template, *{MASK}* stands for a mask space which will be filled by the RL algorithm. *{TOPIC}* will be statically set for each training run to ensure text samples generated are comparable.

Iteration 1: (initial test, only one mask)
Write a {MASK} text about {TOPIC}. Start with general information and end with specific details.

Iteration 2: (add mask at start)
{MASK}. Write a {MASK} text about {TOPIC}. Start with general information and end with specific details. {MASK}

Iteration 3: (add mask at end)
{MASK}. Write a {MASK} text about {TOPIC}. Start with general information and end with specific details. {MASK}.

Iteration 4: (extend text length)
{MASK}. Generate a {MASK} text about {TOPIC}. Start with a {MASK} general description and end with specific details. Write at least 200 words. {MASK}.

Iteration 5 (scientific prompt formulation):
{MASK}. Craft a {MASK} narrative on {TOPIC}. Commence with an introductory overview and conclude with intricate nuances. Ensure a minimum of 250 words. {MASK}.

Iteration 6: (child-friendly prompt formulation)
{MASK}. Make a {MASK} report about {TOPIC}. Start by saying something easy and finish by saying something harder. Your report should be at least 200 words long. {MASK}.

Iteration 7: (showed most promising results)
{MASK}. Shape a {MASK} report concerning {TOPIC}. Start with a general impression and conclude with detailed analyses. The review should encompass at least 200 words. {MASK}.

Appendix C: Base Words for Updated Implementation

ARC: species (Noun), water (Noun), time (Noun), people (Noun), years (Noun), life (Noun), food (Noun), part (Noun), plants (Noun), category (Noun), used (Verb), found (Verb), see (Verb), make (Verb), called (Verb), said (Verb), use (Verb), take (Verb), find (Verb), different (Adjective), same (Adjective), small (Adjective), large (Adjective), high (Adjective), good (Adjective), common (Adjective), important (Adjective), natural (Adjective), little (Adjective)

MCTest: name (Noun), color (Noun), dog (Noun), animal (Noun), boy (Noun), john (Noun), kind (Noun), day (Noun), store (Noun), want (Verb), live (Verb), play (Verb), eat (Verb), put (Verb), favorite (Adjective), many (Adjective), first (Adjective), best (Adjective), new (Adjective), old (Adjective), happy (Adjective), big (Adjective), young (Adjective)

NewsCommentary: countries (Noun), china (Noun), europe (Noun), government (Noun), growth (Noun), country (Noun), america (Noun), become (Verb), made (Verb), come (Verb), remains (Verb), economic (Adjective), political (Adjective), global (Adjective), financial (Adjective), european (Adjective), international (Adjective), public (Adjective), much (Adjective), military (Adjective)

OpenWebText: year (Noun), way (Noun), game (Noun), team (Noun), think (Verb), say (Verb), need (Verb), great (Adjective)

RACE: passage (Noun), author (Noun), writer (Noun), title (Noun), text (Noun), purpose (Noun), statements (Noun), story (Noun), man (Noun), learn (Verb), know (Verb), infer (Verb), tell (Verb), take (Verb), true (Adjective), likely (Adjective), right (Adjective), american (Adjective)

SimpleWiki: user (Noun), party (Noun), football (Noun), city (Noun), state (Noun), movie (Noun), born (Verb), died (Verb), released (Verb), became (Verb), played (Verb), national (Adjective), more (Adjective), local (Adjective), british (Adjective)

SQuAD: type (Noun), century (Noun), war (Noun), world (Noun), known (Verb), considered (Verb), created (Verb), begin (Verb), established (Verb), largest (Adjective), early (Adjective), major (Adjective), french (Adjective)

Bibliography

[1] J. E. Mehan, “Artificial intelligence: Ethical, social and security impacts for the present and the future”. Ely: ITGP, 2022, ISBN: 9781787783737.

[2] L. de Angelis *et al.*, “Chatgpt and the rise of large language models: The new ai-driven infodemic threat in public health”, *Frontiers in public health*, vol. 11, p. 1 166 120, 2023. DOI: 10.3389/fpubh.2023.1166120.

[3] M. U. Hadi *et al.*, “A survey on large language models: Applications, challenges, limitations, and practical usage”, *Authorea Preprints*, 2023. DOI: 10.36227/techrxiv.23589741.v1.

[4] Joachim Roski, Booz Allen Hamilton and Wendy Chapman, “How artificial intelligence is changing health and healthcare: The hope, the hype, the promise, the peril.” In *Washington DC: National Academy of Medicine*, vol. 2019, p. 58.

[5] A. Fernandez, “Artificial intelligence in financial services”, in *Banco de Espana Article 3*, vol. 2019, p. 19. DOI: 10.2139/ssrn.3366846.

[6] S. Moore *et al.*, “Empowering education with llms - the next-gen interface and content generation”, in *Artificial Intelligence in Education*, N. Wang, G. Rebolledo-Mendez, V. Dimitrova, N. Matsuda, and O. C. Santos, Eds., ser. Communications in computer and information science, Cham: Springer Nature Switzerland and Springer, 2023, pp. 32–37, ISBN: 978-3-031-36336-8. DOI: 10.1007/978-3-031-36336-8_4.

[7] A. Malik, S. Mayhew, C. Piech, and K. Bicknell, “From tarzan to tolkien: Controlling the language proficiency level of llms for content generation”, [Online]. Available: http://arxiv.org/pdf/2406.03030v1.

[8] B. AlKhamissi, M. Li, A. Celikyilmaz, M. Diab, and M. Ghazvininejad, “A Review on Language Models as Knowledge Bases”. 2022. [Online]. Available: https://arxiv.org/pdf/2204.06031.

[9] L. Nicolescu and M. T. Tudorache, “Human-computer interaction in customer service: The experience with ai chatbots—a systematic literature review”, *Electronics*, vol. 11, no. 10, p. 1579, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11101579.

[10] S. B. Lee, “Chatbots and communication: The growing role of artificial intelligence in addressing and shaping customer needs”, *Business Communication Research and Practice*, vol. 3, no. 2, pp. 103–111, 2020, ISSN: 2586-5293. DOI: 10.22682/bcrp.2020.3.2.103.

[11] Golam Md Muktadir, “A Brief History of Prompt: Leveraging Language Models.(Through Advanced Prompting)”. 2023. [Online]. Available: https://arxiv.org/pdf/2310.04438.

[12] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, “Prompt engineering in large language models”, in *Data Intelligence and Cognitive Informatics*, I. J. Jacob, S. Piramuthu, and P. Falkowski-Gilski, Eds., ser. Algorithms for Intelligent Systems, Singapore: Springer Nature Singapore and Imprint Springer, 2024, pp. 387–402, ISBN: 978-981-99-7962-2.

[13] Y. Chang *et al.*, “A survey on evaluation of large language models”, *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024, ISSN: 2157-6904. DOI: 10.1145/3641289.

[14] Y. Zhou *et al.*, “Large language models are human-level prompt engineers”, 2023. [Online]. Available: https://arxiv.org/pdf/2211.01910v2.

[15] C. Feng, Y. Sun, K. Li, P. Zhou, J. Lv, and A. Lu, “Genetic Auto-prompt Learning for Pre-trained Code Intelligence Language Models”. 2024. [Online]. Available: https://arxiv.org/pdf/2403.13588.

[16] R. S. Sutton and A. Barto, “Reinforcement learning: An introduction” (Adaptive computation and machine learning), Second edition. Cambridge, Massachusetts and London, England: The MIT Press, 2020, ISBN: 9780262039246.

[17] D. M. Ziegler *et al.*, “Fine-Tuning Language Models from Human Preferences”. 2020. [Online]. Available: https://arxiv.org/pdf/1909.08593v2.

[18] J. D. Chang, K. Brantley, R. Ramamurthy, D. Misra, and W. Sun, “Learning to Generate Better Than Your LLM”. 2023. [Online]. Available: https://arxiv.org/pdf/2306.11816.

[19] T. Amaratunga, “Understanding large language models: Learning their underlying concepts and technologies”. Berkeley, CA: Apress, 2023, ISBN: 9798868800177.

[20] T. Taulli, “Generative AI: How ChatGPT and other AI tools will revolutionize business”. Berkeley, CA: Apress, 2023, ISBN: 9781484293676.

[21] L. S. Lo, “The clear path: A framework for enhancing information literacy through prompt engineering”, *The Journal of Academic Librarianship*, vol. 49, no. 4, p. 102 720, 2023, ISSN: 0099-1333. DOI: 10.1016/j.acalib.2023.102720.

[22] M. Bieswanger and A. Becker, “Introduction to English Linguistics” (utb basics), 2th revised and updated edition. Stuttgart: utb GmbH, 2021, ISBN: 9783838556635.

[23] B. Kortmann, “English Linguistics: Essentials” (Springer eBook Collection), 2nd revised, updated and enlarged edition. Stuttgart: J.B. Metzler Verlag, 2020, ISBN: 9783476056788. DOI: 10.1007/978-3-476-05678-8.

[24] A. Leidinger, R. van Rooij, and E. Shutova, “The language of prompting: What linguistic properties make a prompt successful?” 2023. [Online]. Available: https://arxiv.org/pdf/2311.01967.

[25] Z. Yin, H. Wang, K. Horio, D. Kawahara, and S. Sekine, “Should We Respect LLMs? A Cross-Lingual Study on the Influence of Prompt Politeness on LLM Performance”. 2024. [Online]. Available: https://arxiv.org/pdf/2402.14531.

[26] A. Chapagain, “Hands-on web scraping with Python: Perform advanced scraping operations using various Python libraries and tools such as Selenium, Regex, and others”. Birmingham, UK: Packt, 2019, ISBN: 1789536197.

[27] A. Kilgarriff and C. Yallop, “What’s in a thesaurus?”, in *LREC*, pp. 1371–1379. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2000/pdf/180.pdf.

[28] G. W. Davidson, P. M. Roget, and G. Davidson, Eds., “Roget’s thesaurus of English words and phrases”, 150. anniversary ed. London: Penguin Books, 2003, ISBN: 014051502X.

[29] C. Fellbaum, “WordNet: An electronic lexical database” (Language, speech, and communication), 2. printing. Cambridge, Mass.: MIT Press, 1999, ISBN: 9780262061971.

[30] Z. Guo *et al.*, “Evaluating Large Language Models: A Comprehensive Survey”. 2023. [Online]. Available: https://arxiv.org/pdf/2310.19736.

[31] J. P. Kincaid, J. Fishburne, R. Robert P., C. Richard L., and B. S, “Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel”. Fort Belvoir, VA: Defense Technical Information Center, 1975. DOI: 10.21236/ada006655.

[32] M. Coleman and T. L. Liau, “A computer readability formula designed for machine scoring”, *Journal of Applied Psychology*, vol. 60, no. 2, pp. 283–284, 1975, ISSN: 1939-1854. DOI: 10.1037/h0076540.

[33] J. S. Chall and E. Dale, “Readability revisited: The new Dale-Chall readability formula”. Cambridge, Mass.: Brookline Books, 1995, ISBN: 1571290087.

[34] R. Gunning, “The technique of clear writing”, in *Journal of reading*, vol. 12.8 (1969), pp. 639–646.

[35] G. H. Mc Laughlin, “Smog grading-a new readability formula”, in *Journal of reading*, vol. 12.8 (1969), pp. 639–646.

[36] E. A. Smith and R. J. Senter, “Automated readability index”, in *Aerospace Medical Research Laboratories, Aerospace Medical Division, Air Force Systems Command*, vol. Vol. 66. No. 220.

[37] J. Anderson, “Lix and rix: Variations on a little-known readability index”, in *Journal of Reading*, vol. 26.6 (1983), pp. 490–496.

[38] M. Hu, “The Art of Reinforcement Learning”. Berkeley, CA: Apress, 2023, ISBN: 978-1-4842-9605-9. DOI: 10.1007/978-1-4842-9606-6.

[39] C. J. Watkins and P. Dayan, “Technical note: Q-learning”, *Machine Learning*, vol. 8, no. 3/4, pp. 279–292, 1992, ISSN: 08856125. DOI: 10.1023/A:1022676722315.

[40] N. Ketkar and J. Moolayil, “Deep learning with Python: Learn best practices of deep learning models with PyTorch”, Second edition. New York: Apress, 2021, ISBN: 9781484253632.

[41] L. Ouyang *et al.*, “Training language models to follow instructions with human feedback”. [Online]. Available: http://arxiv.org/pdf/2203.02155.pdf.

[42] M. Deng *et al.*, “Rlprompt: Optimizing discrete text prompts with reinforcement learning”. [Online]. Available: https://arxiv.org/pdf/2205.12548.pdf.

[43] A. S. Tanenbaum and M. van Steen, “Verteilte Systeme: Prinzipien und Paradigmen” (Studium IT), 2nd revised edition. München et al.: Pearson, 2008, ISBN: 9783827372932.

[44] A. Schill and T. Springer, “Verteilte Systeme: Grundlagen und Basistechnologien” (eXamen.press), 2nd edition 2012. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN: 9783642257964.

[45] H. Dowalil, “Modulare Softwarearchitektur: Nachhaltiger Entwurf durch Microservices, Modulithen und SOA 2.0”, 2nd revised edition. München: Hanser, 2020, ISBN: 9783446463776.

[46] H.-Y. Paik, “Web Service Implementation and Composition Techniques” (Springer eBook Collection Computer Science). Cham: Springer, 2017, ISBN: 9783319555423. DOI: 10.1007/978-3-319-55542-3.

[47] S. Patni, “Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS” (SpringerLink Bücher). Santa Clara: Sanjay Patni apress, 2017, ISBN: 9781484226650. DOI: 10.1007/978-1-4842-2665-0.

[48] V. Mnih *et al.*, “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, ISSN: 1476-4687. DOI: 10.1038/nature14236.

[49] “Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality”, 31.03.2024. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/.

[50] L. Zheng *et al.*, “Judging llm-as-a-judge with mt-bench and chatbot arena”, in *Advances in Neural Information Processing Systems*, vol. 36 (2024). [Online]. Available: http://arxiv.org/pdf/2306.05685.pdf.

[51] Aaron Gokaslan and Vanya Cohen, “Openwebtext corpus”, 2019. [Online]. Available: http://Skylion007.github.io/OpenWebTextCorpus.

[52] A. Radford, “Improving language understanding by generative pre-training”. 2018. [Online]. Available: https://hayate-lab.com/wp-content/uploads/2023/05/43372bfa750340059ad87ac8e538c53b.pdf.

[53] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “Squad: 100,000+ questions for machine comprehension of text”. [Online]. Available: http://arxiv.org/pdf/1606.05250.

[54] M. Richardson, C. J. Burges, and E. Renshaw, “Mctest: A challenge dataset for the open-domain machine comprehension of text”, in *Proceedings of the 2013 conference on empirical methods in natural language processing*.

[55] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, “Race: Large-scale reading comprehension dataset from examinations”. [Online]. Available: http://arxiv.org/pdf/1704.04683.

[56] P. Clark *et al.*, “Think you have solved question answering? try arc, the ai2 reasoning challenge”. [Online]. Available: http://arxiv.org/pdf/1803.05457.

[57] V. Mnih *et al.*, “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[58] K. Relan, “Building REST APIs with Flask: Create Python Web Services with MySQL”. Berkeley, CA: Apress L. P, 2019, ISBN: 9781484250228.

[59] M. Grinberg, “Flask web development: Developing web applications with Python”, Second edition. Beijing et al.: O’Reilly, March 2018, ISBN: 9781491991718.

[60] A. Dubey *et al.*, “The Llama 3 Herd of Models”. 2024. [Online]. Available: https://arxiv.org/pdf/2407.21783.

Statutory Declaration in Lieu of an Oath

I – Max Schlosser – do hereby declare in lieu of an oath that I have composed the presented work independently on my own and without any other resources than the ones given.

All thoughts taken directly or indirectly from external sources are correctly acknowledged.

This work has neither been previously submitted to another authority nor has it been published yet.

Mittweida, 05. September 2024 [redacted]

Location, Date Max Schlosser, B.Sc.