Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

Knowledge Graph-Aided Prompt Enrichment: A Study on Large Language Model Cloze-Style Predictions

---

Name: **Daniel Van Cuylenburg**
Student Number: k19012373
Course: Data Science
Word Count: 14,767

Supervisor: Dr Albert Meroño Peñuela

This dissertation is submitted for the degree of MSc in Data Science.

Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr Albert Meroño Peñuela, for his invaluable feedback and support across the project. I would also like to take this opportunity to thank my loving parents for supporting me during my work.

Abstract

In this study, we present a comprehensive analysis of 4 prominent large language models in predicting movie genres, building upon the foundational work of Brate et al. [1]. Our primary focus was to assess the efficacy of enhancing naive prompts with contextual details derived from knowledge graphs to optimize the performance of these models. Drawing from a large dataset of movies with genre labels, the models were assigned a cloze-style (masked token filling) task.

Our rigorous evaluation encompassed 91 distinct prompt styles, encompassing a range of techniques from exploring the most salient knowledge graph properties related to movies, to crafting naturally phrased prompts and employing various paraphrasing strategies. Our findings indicate that the ideal prompt style is intrinsically linked to the model's underlying architecture, its training datasets, and the specific knowledge graph properties and prompt engineering methods applied.

Statistical analysis revealed a significant enhancement in performance when naive prompts were supplemented with knowledge graph properties. Moreover, a subset of the prompts devised in our study demonstrated a statistically significant improvement over the original prompts proposed by Brate et al. [1]. This underscores the pivotal role of prompt engineering in refining the performance of large language models.

Nomenclature

CSV (file)	Comma-separated values
KG	Knowledge graph
LLM	Large language model
NLP	Natural language processing
RDF	Resource description framework
R@n	Recall@n where n = 1, 5, and 10
SF	Significant figures

Contents

List of Figures

List of Tables

1 Introduction

Transformer-based [2] large language models, notably GPT-4 [6], have recently shown to be incredibly powerful at language modelling, achieving state-of-the-art performance on various natural language processing (NLP) tasks, such as passing the legal bar exam [7]. These large language models (LLMs), also referred to as predictive language models, are a type of model designed to comprehend human-like text while considering context. LLMs are trained on vast quantities of textual data, such as Wikipedia pages, large corpora like Reddit posts, news articles, etc., and can learn to predict the subsequent word or phrase in a given sequence of words by conducting syntactic, grammatical, and semantic analysis on a given training string, commonly known as a prompt.

In recent months, LLMs have garnered significant attention, beginning with the release of GPT-3 [8], followed by GPT-4 [6], and later Google's LLM Bard [9], Microsoft's Bing chat LLM [10], and many others. OpenAI's ChatGPT broke the record for the fastest-growing consumer application in history [11][12]. The findings of this study seek to further inform and progress the field of LLMs, focusing intently on the art of prompt engineering.

LLMs, when trained on extensive corpora, possess the capability to generate outputs that are both coherent and contextually pertinent, making them highly useful for a broad range of NLP tasks. They can also recall and reproduce factual data [13][14]. Within the purview of prompt engineering, there is a focus on creating refined sets of instructions for LLMs to ensure the desired quality in generated content [15], usually optimizing the prompt for the best LLM performance. As further elaborated upon in Section 2.3 Related Work, the precision of the LLM's response is largely contingent on its given prompt [1][13][16]. Even the ordering of the same information in a given prompt can be massively influential on an LLM's performance [17][18]. Hence, astute prompt engineering is vital for achieving desired accuracy levels. There has been no widely accepted method that leads to the perfect prompt, with many different methods being proposed, such as using paraphrasing [19][20][21], round-trip translation [22][23][24], or using automatic prompt generators that iteratively find the best-performing prompt in a large search space [25][26].

The primary objective of this project is to examine the efficacy of enhancing cloze-style [27] prompts with information from knowledge graphs to bolster factual prediction accuracy across various LLMs, comparing the performance of each LLM as well as the effectiveness of each enriched prompt style. A cloze-style prompt gets an LLM to predict a masked piece of text in a sentence. For example, for the prompt "Ted is a movie of the genre [MASK].", it is hoped that the LLM would predict the word "comedy" to replace the "[MASK]" token. Central to this study is the utilization of such cloze-style prompts to distil the factual knowledge embedded within the LLMs.

Knowledge graphs (KGs) represent structured depictions of real-world entities (nodes), their attributes, and the relationships between these entities (edges) [28][29][30]. This project aims to retrieve and use KG-derived information about movies - encompassing aspects such as the cast, director, and producer - to bolster prompts fed to LLMs and test if their performance is substantially improved. This project builds upon Brate et al. [1], which had a similar aim and is further discussed in the Section 2.3 Related Work. This study seeks to not only expand upon their methodologies, testing their enriched prompt techniques on a broader array of LLMs, including the two LLMs used in the original paper, but also to improve upon their KG property-inclusive prompt engineering methods. The main aims of the study are listed below, with each objective explained in greater detail in Section 3.1 Functional Requirements:

1. Using existing internet datasets, construct a dataset of movies.
2. Decide which KG attributes to use in the enriched prompts.
3. Implement the enriching prompt techniques discussed in Brate et al. [1].
4. Generate a further set of prompts based on state-of-the-art techniques.
5. Investigate and decide which LLMs this study will evaluate.
6. For each movie, input that movie's list of prompts into each of the LLMs, saving the top 10 most likely predicted words returned.

7. Statistically analyse the results.

Given the growing interest in LLMs and prompt engineering, there is a mounting demand for computer systems capable of processing, understanding, and explaining information about virtually anything. By enriching LLM prompts with KG information and aiming to determine the best prompt style for various tasks, this research paper seeks to contribute to this burgeoning field, fostering the development of more coherent LLMs. More specifically, based on the findings of this paper, constructing LLMs or devising prompts for particular tasks such as movie recommendations, summarizations, genre classification, and numerous other related tasks should become more straightforward. This paper's results will assist researchers and practitioners in optimizing LLMs to generate more accurate and coherent outputs, ultimately enhancing the user experience and expanding the range of potential applications for these powerful language models.

2 Background & Literature Review

This chapter describes the techniques used in this paper to complete my aims. We commence with an in-depth discussion on knowledge graphs, followed by a detailed analysis of LLMs. Subsequently, we delve into a comprehensive review of related work in this domain.

2.1 Knowledge Graphs

As mentioned previously, knowledge graphs (KGs) are organized illustrations of real-world entities (nodes), their characteristics, and the interconnections among these entities (edges) [28][29][30]. KGs were created to consolidate data about the world into a consistent, computer-readable format, allowing for more efficient information storage, parsing, retrieval, and reasoning.

A host of strategies has been employed for the organization of data within KGs. One such prominent method is the Resource Description Framework (RDF), which employs 'triples' - the fundamental units of information in KGs. Each triple comprises three constituents: a subject, a predicate, and an object [31], or alternatively an entity, relationship, and value. The RDF model is recognized as the benchmark for data exchange across the web [32]. The subject represents the central entity under discussion, typically characterized by a unique identifier like a URI (Uniform Resource Identifier). The predicate, or the property, delineates the nature of the relationship between the subject and the object. The object signifies the entity to which the subject is linked. The object may represent another entity (subsequently identified by a unique identifier) or be a literal value, such as a numerical figure or a textual string.

The next subsections describe popular KGs currently used worldwide that were considered for this study.

2.1.1 YAGO

YAGO, an acronym for Yet Another Great Ontology, is a large-scale ontology known for its comprehensive coverage and precision. Comprising more than 2 billion type-consistent RDF triples for 64 million entities, YAGO was automatically derived from sources such as Wikipedia, WordNet, and GeoNames in 2007[3][33][34], making it one of the world's largest KGs. Each fact in YAGO undergoes a pipeline of filtering, constraint checking, and de-duplication, resulting in a manually verified accuracy exceeding 95%. However, YAGO's reliance on Wikipedia infoboxes as one of its primary sources has curtailed its popularity relative to Freebase[4] and Wikidata, both of which accommodate a broader spectrum of data types and exploit a wider range of sources [35].

2.1.2 DBpedia

DBpedia[5], one of the most widely used semantic databases, is dedicated to extracting structured content from the information generated in Wikipedia, with its downloads exceeding 600 thousand files per year. Information is classified into categories such as places, people, creative works (books, movies, etc), and so forth. For each category, a set of properties exists that describe instances of that category. To date, the dataset describes 228 million entities. Unlike some other KGs, DBpedia adheres to a release cycle for updates. DBpedia's data can be accessed freely, most commonly via SPARQL queries.

SPARQL, standing for SPARQL Protocol and RDF Query Language, is a semantic query language used to retrieve information from databases, facilitating the extraction of data stored in the RDF triples format [31]. SPARQL allows users to formulate complex, flexible queries across RDF models.

---

[3] https://yago-knowledge.org/

[4] Freebase was a large collaborative KG that ceased operation in 2014, after which Microsoft, the company owning Freebase, migrated its data to Wikidata.

[5] https://www.dbpedia.org/

[6] https://en.wikibooks.org/wiki/SPARQL/WIKIDATA_Qualifiers,_References_and_Ranks#media/File\protect\protect\leavevmode@ifvmode\kern+.2222em\relaxDatamodel_in_Wikidata.svg

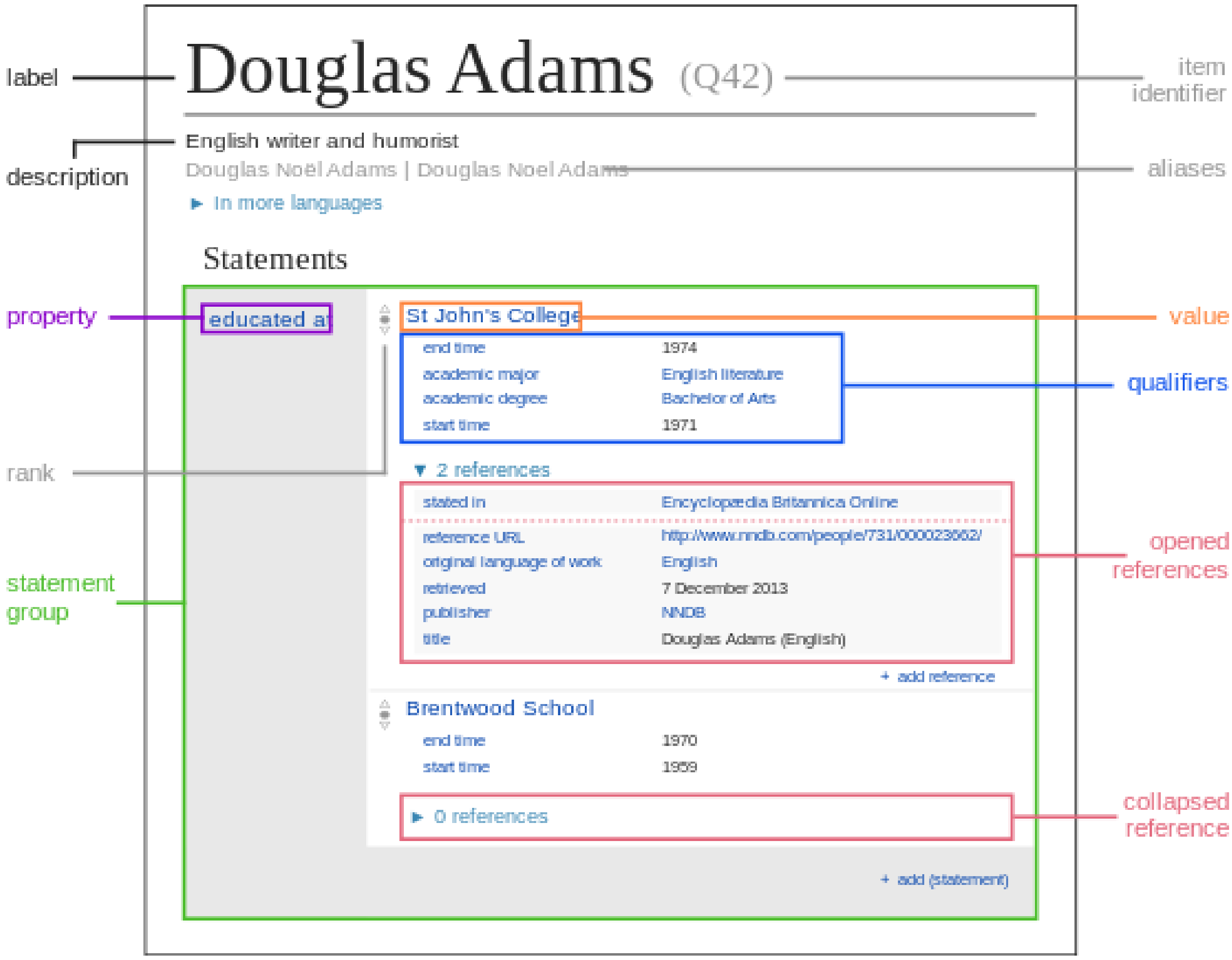

Figure 1: Wikidata data model example for Douglas Adams. [6]

2.1.3 Wikidata

The Wikidata KG[7][36][37] is a collaborative, multilingual, universally accessible knowledge database that collects data about virtually anything, including people, events, places, concepts, and objects. Initiated by the Wikimedia Foundation in 2012, which also manages Wikipedia, Wikidata functions as a central repository for structured data originating from Wikipedia - one of the largest and most frequently cited online resources, serving as a free encyclopedia - as well as other Wikimedia projects.

As mentioned previously, Wikidata uses the RDF format, which is a flexible, graph-based data model used to represent information; each RDF triple consists of a subject, predicate, and object. Each item in Wikidata, which represents the subject in an RDF triples model, is identified using a unique string, which is the letter 'Q' followed by a numerical sequence [8]. Each item represents a concept or object in the real world. For example, "London" is represented by the identifier "Q84"[9], while the "City of London" is represented by "Q23311"[10]. Despite initial impressions suggesting these items should be consolidated, Wikidata strives to accurately represent each unique entity: "London" refers to the capital city of the UK, whereas the "City of London" denotes a county located centrally in London, spanning merely 1.12 square miles. This example illustrates how Wikidata aims to comprehensively and accurately represent all unique entities, no matter how confusing or similar they may seem. Consequently, Wikidata's robust structure makes it an apt choice for the KG in this paper, as film titles often correspond to separate entities (for instance, "Lincoln" or "Selma"), and Wikidata is designed to differentiate these entities into unique items.

[7] https://www.wikidata.org/wiki/Wikidata:Main_Page
[8] https://www.wikidata.org/wiki/Help:Items
[9] https://www.wikidata.org/wiki/Q84
[10] https://www.wikidata.org/wiki/Q23311

Properties in Wikidata represent the predicates. These are identified by the letter 'P' followed by a numerical sequence[11]. For example, a frequently utilized property is "P31", which represents an "instance of"[12]. Objects, referred to as values in Wikidata, can be represented similarly to items, using the letter 'Q' followed by a numerical sequence when referencing another item. However, they can also adopt simple formats such as a string, an integer, or a date, among others[13]. For instance, one of London's properties, "continent", is denoted by "P30", while its corresponding value, "Europe", is signified by "Q46"[14].

Beyond these basic components, Wikidata statements can incorporate references, qualifiers, and ranks for enhanced detail, but these are not used in this study. In summary, a Wikidata statement forms a detailed assertion in the form Item - Property - Value. Figure 1 provides an illustrative example of the Wikidata data model. Wikidata also features a SPARQL query API[15], enabling users to access and extract data stored in Wikidata via the SPARQL query language. Users have the option to either input their SPARQL queries directly into the website or utilize Wikidata's API service.

2.2 Large Language Models

Large Language Models (LLMs), particularly in recent years, have surged in popularity, playing a significant role in the latest advances in natural language processing (NLP) [38][39][40]. LLMs boast a wide array of applications, including text summarization, text translation between languages, question answering, semantic analysis, and providing explanations for text. Recent advancements in NLP, such as the introduction of Transformer-based models [2], like BERT [41] and RoBERTa large [42], the LLMs utilized in Brate et al. [1], as well as newer models like GPT-4 [6], have shown a marked enhancement in performance compared to previous LLMs. Transformer-based [2] models represent state-of-the-art neural network architectures, and are discussed in detail in Section 2.2.1 The Transformer Architecture. LLMs like BERT [41], at least for text prediction tasks, essentially compute the probability of each word (or each character) in their vocabulary occurring at the given position that needs to be filled, selecting the words with the highest probabilities.

Without an appropriate prompt for a given task, any LLM is rendered useless. Prompt engineering involves crafting a set of instructions for an LLM, designed to enforce rules, automate processes, and ensure specific qualities (or quantities) of a generated output [15]. Task performance relies heavily on the quality of a given prompt, and the most effective prompts are created by either humans or automated prompt generators [25][26], depending on the task.

Zero-shot learning, a subfield of transfer learning, occurs when an LLM is configured to execute tasks not presented during its training period [43][44]. This is facilitated by capitalizing on the LLM's comprehension of other pertinent tasks, utilizing elements such as attributes or textual descriptions to forge linkages between familiar and unfamiliar categories. For example, given the knowledge that "a zebra looks like a horse with stripes", a child who has never seen a zebra before would be able to recognize one, assuming that they know what a horse looks like and what a striped pattern looks like [45]. Conversely, few-shot learning manifests when an LLM is supplemented with minimal training data, typically just a handful of instances, for a novel task or class [46]. An example of few-shot learning could be a child that is able to learn multiplication based on prior knowledge and given a few examples ($2 \times 3 = 2 + 2 + 2,\ 1 \times 3 = 1 + 1 + 1$) [46]. This study focuses on zero-shot learning methodologies.

The next subsections describe each of the selected LLMs for this study. After careful deliberation, decisions were reached regarding which LLMs to incorporate in this study, with the selection criteria and justifications being outlined in Section 3.3.2 Large Language Model Selection. Table 1 outlines all pertinent information concerning each selected LLM and its respective research paper, while Table 2 displays all the available GLUE benchmark scores for each of the selected LLMs. The General Language Understanding Evaluation (GLUE) benchmark [47] is a popular set of 9 NLP tasks used to evaluate the performance of LLMs. As previously stated, the LLMs to be analyzed include the two LLMs from the original paper Brate et al. [1], specifically BERT [41] and RoBERTa large [42]. All of the LLMs in this study use the Transformer architecture [2], which is initially described in the next subsection.

[11] `https://www.wikidata.org/wiki/Help:Properties`
[12] `https://www.wikidata.org/wiki/Property:P31`
[13] `https://www.wikidata.org/wiki/Help:Statements#Values`
[14] `https://www.wikidata.org/wiki/Q46`
[15] `https://query.wikidata.org/sparql`

LLM	Company	Creation Year	Number of Parameters	Paper	Training Datasets
BERT	Google	2018	110 million	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [41]	BooksCorpus [48], English Wikipedia
RoBERTa Large	Facebook	2019	354 million	RoBERTa: A Robustly Optimized BERT Pretraining Approach [42]	BooksCorpus [48], English Wikipedia, CC-News [49], OpenWebText [50], STORIES [51]
BART Large	Facebook	2020	Unknown	BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [3]	BooksCorpus [48], English Wikipedia, CC-News [49], OpenWebText [50], STORIES [51]
ALBERT Large v2	Google Research	2019	18 million	ALBERT: A Lite BERT for Self-supervised Learning of Language Representations [52]	BooksCorpus [48], English Wikipedia

Table 1: LLM details, taken from either the LLMs individual papers or from Hugging Face[16].

LLM	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
BERT[41]	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.60
RoBERTa large[53]	90.2/90.2	92.2	94.7	96.4	68.0	92.4	90.9	86.6	89.10
BART large[54]	-	-	-	-	-	-	-	-	87.05
ALBERT large v2	-	-	-	-	-	-	-	-	-

Table 2: GLUE benchmark tasks for different LLMs. Note that some test scores for BART Large and ALBERT Large v2 could not be found, although the average test score for BART Large was found. The sources of the GLUE scores are presented in the LLM column.

2.2.1 The Transformer Architecture

The Transformer has emerged as the most proficient neural network architecture for neural language modelling [53] since its inception in Vaswani et al. [2] in 2017. Figure 2 illustrates the Transformer-model architecture. The operational principles of Transformers are described in the following paragraphs.

Initially, any input is tokenized and transformed into continuous vectors which are fed into the model. So as to incorporate the order of the words, additional positional encodings are added to these input embeddings. Each of these inputs then goes through the self-attention mechanism, which computes a score for each word in the input to assess its importance. These scores indicate how much focus should be put on each word in the sequence. The final output of the self-attention layer is the embedded vectors weighted by these attention scores.

The primary role of the encoder is to understand the input data and compress it into an abstract, yet comprehensive representation that the decoder can use. The encoder takes the input vectors and runs the self-attention mechanism multiple times in parallel to find different contextual relationships between tokens in the input data. After each self-attention phase, each of the representations goes through a feed-forward neural network. The outputs of each encoder layer serve as inputs for the subsequent encoder layers until the final layer is reached. This output is then passed to the decoder, which, similarly to the encoder, also has several identical layers.

The decoder's role is to generate the output data from the encoded input. Only 1 of the 4 LLMs used

[16]`https://huggingface.co/transformers/v2.4.0/pretrained_models.html`

in this paper uses the traditional decoder from Vaswani et al. [2], so the decoder is not described in great detail here. The output of the decoder layers is finally put through a linear layer followed by a softmax to generate a prediction. A softmax function essentially transforms a vector of real numbers into a probability distribution.

In summary, the Transformer architecture takes in a sequence of tokens, applies self-attention to each token by considering all tokens in the sequence, uses the attention scores to form a context-aware representation of each token, and uses this to generate an output (like a translation, a classification, and so on). BERT has 12 Transformer layers, while all of the other LLMs used in this study have 24 layers.

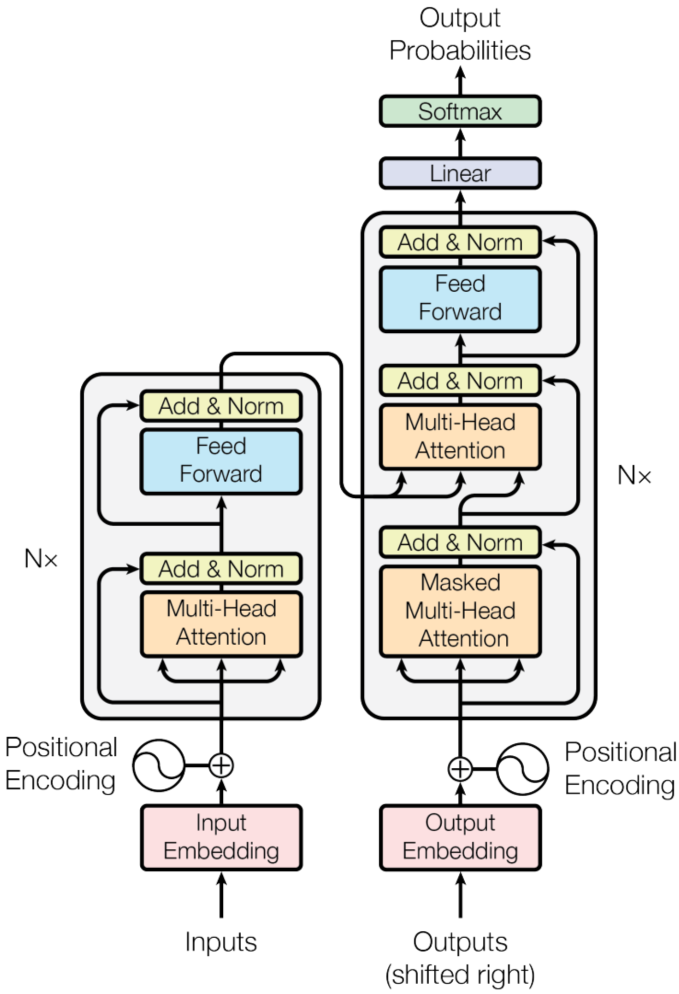

Figure 2: Schematic representation of the Transformer-model architecture as presented by Vaswani et al. [2]

2.2.2 BERT

BERT [41], an acronym for Bidirectional Encoder Representations from Transformers, is an LLM devised by Google in 2018 that employs the Transformer [2] architecture. Unlike traditional LLMs that process text unidirectionally, BERT is bidirectional, meaning that it attempts to understand the context on both the left- and right-hand side of a given token in the same layer. BERT consists of a stack of Transformer encoders. As in the Transformer model, BERT's inputs are tokenized, vectorized, and passed through the Transformer neural network. BERT is implemented in two main steps: pre-training, and fine-tuning.

Pre-training BERT involves using two unsupervised learning tasks. The initial task, referred to as masked language modelling, is also known as a cloze task [27]. This involves masking some percentage of the input tokens at random, followed by predicting those tokens at a cross-entropy loss using the words surrounding the masked tokens. Cross-entropy loss measures the degree to which the predicted probability distribution aligns with the true distribution. The hidden vectors corresponding to the masked tokens are fed into a softmax function over the LLM's vocabulary. For BERT, the training data generator randomly selects 15% of the token positions for prediction. Out of these selected tokens, each token is substituted with "[MASK]" 80% of the time, with a random token 10% of the time, and remains unchanged 10% of the time. If the [MASK] token was employed 100% of the time, a mismatch would occur between the pre-training and fine-tuning steps, as the [MASK] token would not appear during the fine-tuning step.

The second pre-training task, called next sentence prediction, is used to train BERT to understand sentence relationships. This task can be performed using any text corpus, but Devlin et al. [41] used the BookCorpus [48], containing 800 million words, and English Wikipedia[17] passages, containing 2.5 billion words. Specifically, this task involves partitioning a text corpus into examples, where each example consists of a sentence succeeded by the actual next sentence 50% of the time, and by a random sentence from the corpus the remaining 50% of the time. BERT is trained to predict the subsequent sentence and achieves an accuracy exceeding 97% in this task.

The fine-tuning of BERT constitutes the second step, enabling BERT to model various downstream tasks. For each task, the inputs and outputs are inserted into BERT, upon which all of BERT's parameters undergo fine-tuning. This allows BERT to learn features and representations specific to the training datasets. By adjusting all the parameters, the model can more effectively adapt to the unique characteristics and nuances of the given tasks. Some examples of tasks employed in this step encompass question-answering, sentiment analysis labelling, and so forth.

2.2.3 RoBERTa Large

RoBERTa large [42], short for Robustly optimized BERT approach, is a BERT variant developed by Facebook in 2019. Several modifications were made in RoBERTa compared to BERT: "(1) training the model longer, with bigger batches, over more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data". Changes (1) and (3) are relatively self-explanatory. Change (2) was implemented after Facebook discovered that "removing the next sentence prediction loss matches or slightly improves downstream task performance", a finding contradicting Devlin et al. [41]. As for step (4), Facebook noted that BERT's static approach to masked language modelling resulted in a single static mask, since this step was only performed once during data preprocessing. Facebook introduced dynamic masking, where a masking pattern is generated each time a sequence is fed into the model. This technique has been found to perform comparably or slightly better than static masking, and has therefore been used for RoBERTa.

As well as using the BookCorpus [48] and English Wikipedia like BERT, Facebook also used three more corpora, those being CC-News [49], a collection of over 44 million English documents made up of news articles from all over the world collected between September 2016 and March 2018; the OpenWebText corpus [50], which is an open-source recreation of the WebText corpus created in Radford et al. [55] containing millions of webpages scraped from URLs in Reddit comments that had more than 2 upvotes; and STORIES [51], a corpus containing a subset of CommonCrawl[18] data filtered to match the story-like style of Winograd schemas. A Winograd schema is a pair of sentences that differ in only a few words that contain referential ambiguity that is resolved in opposite directions in the pair of sentences [4]. Levesque et al. [4] gives an example of this, presented in Table 3. These schemas are used to test the abilities of LLMs at handling coreference resolution (linking pronouns to correct nouns), leveraging real-world knowledge, and understanding natural language in a human-like way.

Sentence	**Correct Answer**
The trophy doesn't fit in the brown suitcase because it's too big. What is too big?	the trophy
The trophy doesn't fit in the brown suitcase because it's too small. What is too small?	the suitcase

Table 3: Example of the Winograd schema as described by Levesque et al. [4]

[17] `https://www.wikipedia.org/`
[18] `https://commoncrawl.org/`

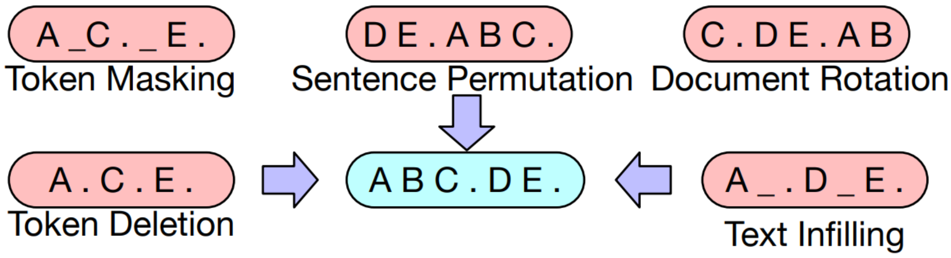

Figure 3: Transformations for noising BART's inputs. Figure taken from Lewis et al. [3]

2.2.4 BART Large

BART large [3], an acronym for Bidirectional and Auto-Regressive Transformers, is another Transformer-based LLM, introduced by Facebook in 2020. BART is a denoising autoencoder that operates using a sequence-to-sequence model. In contrast to the other LLMs included in this paper, BART makes use of both the encoder and decoder aspects of the Transformer architecture. The decoder accepts the encoder's output and produces an output sequence in an auto-regressive manner. This means that one token is generated at a time based on a probability distribution, using the previously generated tokens as additional input when generating the next token, maximizing the likelihood of a word given its previous words.

The pre-training process of BART unfolds in the following manner. Like BERT, random tokens are replaced with "[MASK]" tokens. Following this, random tokens are deleted from the input; unlike token masking, BART must decide which positions are missing inputs. Another pre-training step involves text infilling, where a number of text spans, with span lengths taken from a Poisson distribution, are replaced with a single "[MASK]" token. It is noted that this includes 0-length text spans. This technique teaches BART how to predict the number of tokens missing from a text span. Further techniques involve randomly shuffling the order of sentences in a document (sentence permutation), or rotating a document around a randomly selected token. These techniques teach BART to identify the start of a document. Figure 3 illustrates the transformations BART performs for noising the input.

To put it differently, while BERT learns to complete gaps in a text (akin to filling in blanks within a sentence - a cloze task), BART learns to rectify corrupted text (similar to editing a sentence with errors). Moreover, BART Large's training utilized the same training corpora as RoBERTa.

2.2.5 ALBERT Large v2

ALBERT large v2 [52], which stands for A Lite BERT, is a model developed by Google Research that improves on BERT by including various optimizations, primarily concerning memory usage, model size, and training time. Despite having fewer parameters, ALBERT often achieves similar or even better performance than BERT on some tasks. This model has been included in the study because of its potential to offer intriguing insights into the performance of an optimized but compact version of BERT on our cloze-style task. ALBERT large v2 was also trained on the same corpora as BERT.

Hugging Face, the source of the LLMs, described in Section 3.3.2 Large Language Model Selection, does not provide a reliable model for ALBERT large. Furthermore, since ALBERT was trained using fewer parameters than all of the other LLMs, this indicates that it may perform worse than all of the other LLMs depending on the specific task. As a result, ALBERT large v2 has been used in this paper, which is a further optimized version of ALBERT large, and seems to be the version of ALBERT that is most used by the LLM community[19], hopefully allowing the results of this paper to be more applicable to this field of work.

2.2.6 Other Large Language Models

The Helsinki MarianMT[20] LLM is used for the construction of some of the prompts, where we implemented a paraphrasing technique that involves translating a prompt to and from a foreign language, further discussed

[19] ALBERT large v2, as of July 2023, has over 6,000 downloads, while ALBERT large v1 has just under 500 downloads.
[20] `https://huggingface.co/docs/transformers/model_doc/marian`

in Section 4.3 Prompt Engineering. This LLM uses Marian [56], which is an efficient neural machine translation framework that was developed by Microsoft. Marian also uses the Transformer architecture [2], trained on parallel sentences in the source and target languages.

FLAN-T5 [57], which stands for "fine-tuning language models Text-to-Text Transfer Transformer", is an LLM that was released by Google in 2022 and is also used for the construction of some of the prompts in this study. This LLM is a fine-tuned version of the T5 LLM [58], which is an LLM that treats every NLP task as a text-to-text problem, showing impressive performance at these tasks. Specifically, this LLM is used to rephrase some of the best-performing prompts, further discussed in Section 4.3 Prompt Engineering.

DeBERTa large [53], ELECTRA large [59], and XLNet large [60] were 3 other LLMs that showed better performance than BERT at various benchmarks tests like the GLUE benchmark [47], and that were initially used in this study. However, they were deemed unsuitable after they were unable to predict any movie genres, regardless of the prompt style.

2.3 Related Work

This study is mainly inspired by two papers. Penha et al. [61] investigated whether BERT was able to provide sufficient recommendations for books, movies, and music, specifically in conversational settings, without any explicit fine-tuning or training for this type of task. The study found that although BERT has knowledge stored in its parameters about the content of movies, books, and music, and although BERT shows some potential for recommending media, BERT fails on conversational recommendation when faced with adversarial data. Various text prediction tasks were given to BERT in the form of prompts, where some information is given before asking BERT to fill in a word or phrase. These included filling in the genre of a piece of media, giving a description of a piece of media and filling in the name of that piece of media, and filling in a recommendation for a user based on that user's previously liked media.

Brate et al. [1] explores the idea of improving the performance of LLMs, specifically, BERT [41] and RoBERTa large [42], two Transformer-based models, leveraging relevant information from KGs in their prompts so as to provide the LLM with more explicit context, improving the LLM's ability to generate factually correct predictions. This study aims to expand on these two papers, incorporating a wider variety of LLMs and expanding upon the enriched prompting techniques.

As mentioned in Penha et al. [61], their prompt styles are originally based on two papers specifically. Petroni et al. [16] explores the impact of adding contextual information into prompts, concluding that contextual information substantially improves BERT's zero-shot cloze-style question-answering performance. Rocktaschel et al. [13] also makes use of cloze-style prompts, to evaluate the factual and commonsense knowledge available through BERT.

Other studies have also explored the use of context within LLM prompts. Liu et al. [62] provides a great comprehensive overview of the use of prompts for the completion of NLP tasks using LLMs like GPT-3 [8] and BERT [41]. This paper discusses the process of crafting effective prompts, discussing the differences between prompt styles, such as the use of prompt templates. Manually designed prompt templates have been used in the previously mentioned Rocktaschel et al. [13] and Brown et al. [8].

Studies have also shown that rewording a prompt so that it is expressed in natural language can also improve the performance of LLMs. Denny et al. [63] showed that, out of all of their programming problems that were not initially solved by GitHub Copilot[21], a popular programming LLM [64], rewording these prompts in natural language allowed Copilot to solve 60.9% of these problems. Ruis et al [65] finds that, out of the 11 LLMs that they tested, 7 of the LLMs performed better with the naturally worded prompts on average compared to the structured prompts when testing how well the LLMs understand the implicit semantic meaning of the utterances.

Liu et al. [62] also surveys the use of automatic prompts, which are prompts that search for templates described in a discreet space, often corresponding to natural language phrases. Yuan et al. [19] proposes paraphrasing manually devised prompts by using synonyms from a thesaurus[22] to narrow the search space of potential prompts. Other papers have also found increased performance from prompts that were paraphrased [20][21]. Jiang et al. [22] proposes a round-trip translation of a prompt, where a prompt is translated into

[21]https://github.com/features/copilot

[22]https://www.wordhippo.com/

another language and then back to English to express the same meaning of the prompt in different words. This same paper also proposes mining-based and paraphrasing-based methods of automatically generating high-quality, diverse prompts. Other papers have also found success with round-trip translation [23] [24]. Both of these automatic methods have been incorporated into this study.

Haviv et al. [25] proposes a prompt rewriter specifically optimized to improve the performance of BERT, aiming to bridge the gap between natural language prompts and the implicit language of BERT. Zhou et al. [26] similarly proposes an "Automatic Prompt Engineer" for automatic instruction generation and selection, maximizing a chosen score function to select an optimal prompt; these automatically generated prompts outperform the prior LLM baseline by quite a large margin, achieving either comparable or better performance to the instructions generated by human annotators. Note that, as discussed in some of these papers, some of these automatic approaches may not be considered true zero-shot or few-shot learning, as a large amount of annotated data may be required to automatically generate the initial prompts.

When constructing prompts, another thing to keep in mind is that the ordering of the content in a prompt can massively affect an LLM's prediction accuracy. Lu et al. [17] showed that the order of few-shot prompts can make the difference between an LLM predicting at a state-of-the-art level compared to randomly. On the other hand, although Pham et al. [18] does show that shuffling the order of words in NLP tasks does lower the accuracy of BERT's performance with GLUE benchmark [47] tests, the change is much less subtle than one would expect. As seen in Brate et al. [1], as well as Petroni et al. [16], adding more contextual information does not always yield a more accurate result; the prompts in Brate et al. [1] with all of the contextual information added were not the highest performing prompts. It is therefore imperative that we experiment with various KG property orders in this study.

Other papers have attempted to utilize KGs to (pre-)train LLMs (rather than using the KG information in prompts) for better performance. Zhang et al. [66] created an LLM that incorporated structured KG information to significantly improve the performance of BERT on common NLP tasks. ERNIE [66], another LLM, can make use of lexical, syntactic, and KG information simultaneously. He et al. [67] attempts to incorporate both KG relationships, as well as entities, into a training process to obtain a KG-enhanced pre-trained LLM named KLMo. Results suggested that KLMo achieved great improvements on several specific knowledge-driven tasks, such as relation classification and entity typing, compared to other state-of-the-art LLMs similar to BERT. KG information has successfully been incorporated into the pre-training processes of LLMs to achieve better performance, suggesting that KG information can certainly be utilized in multiple other ways apart from the prompt enriching techniques attempted in this paper.

In the next chapter, we will discuss the functional requirements and technical specifications of this study.

3 Specification Requirements

3.1 Functional Requirements

Each of this study's aims is broken down into specific functional requirements.

1. Using existing internet datasets, construct a dataset of movies.
   (a) Using the ML-25M (MovieLens 25 Million) dataset [5], a list of movies must be downloaded and saved as a CSV file.
2. Decide which KG attributes to use in the enriched prompts.
   (a) For each movie, all of the KG properties that were used in Brate et al. [1] must be downloaded - these are displayed in Table 6.
   (b) Movies with missing data in any of these properties must be removed.
3. Implement the enriching prompt techniques discussed in Brate et al. [1].
   (a) For each movie, construct 19 separate prompts. This includes one unenriched prompt, 9 enriched prompts separated by the word "and", and 9 enriched prompts separated by commas instead. An example of a set of these prompts can be seen in Table 7.
4. Generate a further set of prompts based on state-of-the-art techniques.
   (a) After letting all of the LLMs process all of the prompts in the style of Brate et al. [1], investigate which properties led to the best LLM performances. Based on these results, add a further set of prompts constructed based on these well-performing properties, with the goal of achieving higher recall scores[23] than Brate et al. [1]. An example of these prompts can also be seen in Tables 7, 8, and 21.
   (b) In the development of further prompts, this study is focused on genuine zero-shot learning. Therefore, techniques that evaluate and iteratively enhance a prompt's performance cannot be utilized. Any such methodologies would compromise the pure zero-shot learning approach we are aiming for.
5. Investigate and decide which LLMs this study will evaluate.
   (a) BERT [41] and RoBERTa large [42] must be included in the final list of LLMs, as these were the two LLMs used in Brate et al. [1].
   (b) A list of other LLMs must be decided upon (further described in Section 3.3.2 Large Language Model Selection).
6. For each movie, input that movie's list of prompts into each of the LLMs, saving the top 10 most likely predicted words that are returned.
   (a) Each generated clozed-styled prompt must be processed by each LLM.
   (b) The responses for each individual LLM and prompt style must be saved in separate CSV files for each LLM.
7. Statistically analyse the results.
   (a) This study's results must be compared with the results of Brate et al. [1], attempting to validate their findings. Any matching results/discrepancies must be documented.
   (b) The accuracy of each prompt style for each LLM across the whole movie dataset must be calculated.

[23]Recall scores are explained in detail at the start of Section 5 Results.

(c) Paired t-tests must be performed between the unenriched prompt style[24] and all of the enriched prompt styles for each LLM.

(d) Paired t-tests must also be performed between the best-performing prompt styles from Brate et al. [1] and this study's best-performing prompt styles for each LLM.

(e) A list of movies that were classified correctly the highest number of times on average must be produced.

3.2 Non-Functional Requirements

Several non-functional requirements have been adhered to throughout the study.

1. The software must be easy to deploy, configure, and maintain in the future.
2. The results produced through the methodologies described in this paper must be reproducible on other machines.
3. The software must not collect any sensitive, identifiable data[25].
4. The software produced should meet the highest professional and ethical standards set out by the British Computer Society.

3.3 Technical Specifications

The collection, cleaning, and formatting of the dataset encompassing books and their KG attributes will be carried out using Python, with the resultant output stored as CSV files. The subsequent processes, which involve constructing prompts and feeding them to each of the chosen LLMs, will likewise be executed in Python. These choices are predicated on the fact that Python is widely recognized as the de facto standard for data processing and is the most commonly utilized programming language for data analysis [68] [69]. In addition, the coding practices prescribed in the Google Python Style Guide[26] will be adhered to.

All of the translation, paraphrasing, and mask-filling will be performed on a desktop computer with 32GB of RAM, using an Asus DUAL GeForce GTX 1060 3GB graphics card. The next two subsections describe the selection methodology for the KG and the LLMs.

3.3.1 Knowledge Graph Selection

Numerous established KGs are available, with some of the most popular KGs already being described in Section 2.1 Knowledge Graphs. For this study, it is important to select a KG that has as much movie property information as possible available, while also being reliable and easily accessible.

For this research, the Wikidata KG has been selected to extract movie properties. As previously described, Wikidata utilizes the RDF format supplemented by additional elements for its data structuring, allowing users to retrieve this RDF formatted information via SPARQL. This is quite convenient for this study. This choice was also predicated on its consistent updates from a dedicated community of editors, coupled with its relative reliability, substantiated by authoritative sources[27]. In practice, this implies that all statements added to Wikidata should include a reference. Similar to other Wikimedia projects, Wikidata is governed by a cohort of unpaid contributors, collaborating to maintain and augment the existing data when required. These contributors abide by a set of guidelines and policies, which are themselves constructed by the community.

[24] The unenriched prompt style contains no KG movie properties, only the movie's title.

[25] For example, the Wikidata KG editor information

[26] `https://google.github.io/styleguide/pyguide.html`

[27] `https://www.wikidata.org/wiki/Wikidata:Verifiability`

3.3.2 Large Language Model Selection

In selecting the LLMs for this study, multiple factors were considered. Given that thousands of popular LLMs have emerged over merely the past five years[28], it was crucial to delineate precise and coherent criteria for LLM selection.

Primarily, any chosen LLM had to have some form of published documentation associated with it, such as a research paper. This stipulation was necessary because the use of an undocumented LLM would prevent any comprehensible interpretation of performance variations amongst models. Research papers pertaining to each model describe both the operational intricacies of the respective LLM as well as the specific training data upon which each LLM was trained. This information is crucial for the interpretation of the results. Moreover, such papers typically showcase the performance of the LLM across a range of benchmark tests and often compare the LLM to contemporaneous state-of-the-art models, which can further aid in discerning performance differences. Secondly and similarly, each chosen LLM had to be at least moderately renowned within the LLM community. There would be little utility in assessing niche LLMs with less than a thousand downloads, unseen and likely to remain overlooked by the LLM community.

Thirdly, every selected LLM must exhibit performance that is at least comparable to BERT, the least effective LLM according to the GLUE benchmark [47]. This prerequisite was relatively unproblematic, as all of the LLMs included in this study had already compared their performance against BERT in their original papers, demonstrating superior results in a large part if not all of the NLP tasks.

Having established these foundational conditions, additional LLM features were also evaluated. This study necessitates significant changes in each LLM's underlying architecture for an LLM to be considered for inclusion. Absent this condition, the study would employ multiple LLMs sharing the same underlying architecture, with the only variance being the distinct corpora upon which they were trained/fine-tuned. It should be noted that if an LLM had a "large" version available, which entails the same LLM trained on a more extensive corpus, also including more Transformer layers, which invariably performs superiorly on NLP tasks, then that larger LLM was selected in place of the original. Only BERT is an exception to this, following Brate et al.'s methodology [1]. Consequently, we've used the standard BERT in our study, consistent with their approach. Other variations of these LLMs, such as "roberta-xlarge" or "albert-xlarge", were also excluded from this study for analogous reasons. Moreover, a comparison between BART large and RoBERTa extra large would be inherently unfair due to the discrepancies described.

As mentioned previously, each of this study's chosen LLMs have been documented in Section 2.2 Large Language Models. Each of these LLMs meet the conditions outlined in this section. All of these LLMs are available through Hugging Face[29], who describe themselves as "on a journey to advance and democratize artificial intelligence through open source and open science". Hugging Face is known for developing tools for machine learning applications, most notably its Transformer [2] models library that caters to a wide variety of NLP tasks. These models are easily shared through the website to millions of people - the most downloaded fill-mask model available on the website is BERT [41], which, as of June 2023, has over 50 million downloads[30].

In the next chapter, we will talk about how we implemented the functional requirements discussed in this chapter.

[28] For instance, Hugging Face, which is described towards the end of this section, contains thousands of models with more than one thousand downloads each.

[29] `https://huggingface.co/models?pipeline_tag=fill-mask&language=en&sort=downloads`

[30] `https://huggingface.co/bert-base-uncased`

4 Methodology & Implementation

This chapter documents the implementation of the requirements delineated in Section 3.1 Functional Requirements which comprise 4 main stages. The first stage involved the acquisition and preprocessing of the ML-25 [5] dataset. Subsequently, the KG properties pertinent to this study were specified and retrieved from Wikidata. The third stage saw the organization of these KG properties into a set of 91 distinct prompts. Finally, these prompts were fed into the chosen LLMs. Table 4 presents the Python files, their corresponding functionalities, and their output folders. Figure 4 illustrates our whole methodology before the statistical analysis of the results.

4.1 Dataset Retrieval and Cleaning

Brate et al. [1] used the ML-25M (MovieLens 25 Million) [5] dataset, which contains over 25 million ratings across 62,423 movies, with each movie being tagged with zero or more genres from a possible list displayed in Table 5. To validate the findings of Brate et al. [1], the present study also leveraged this dataset. For the purpose of reproducibility, the dataset was directly procured from its official repository[31].

Upon acquisition, the dataset underwent rigorous preprocessing. Any redundant data not used in this study is discarded, notably the rating metrics. The remaining columns were "movieId", "imdbId", "tmdbId", "title", and "genres". Interestingly, different LLMs have varying outputs regarding punctuation. To maintain uniformity, any word predictions that encompassed punctuation were ignored. As an adjustment, the "film-noir" genre was rebranded as "noir", while "sci-fi" was excluded. Furthermore, all genre descriptors were transformed to lowercase. Movies with no genre labels were eliminated, considering their irrelevance to the study.

Python File Name	**Purpose**	**Discussed in**	**Output Folder: Data/...**
fetch_movies_kg	Retrieves KG properties	Section 4.2 Knowledge Graph Querying	Dataset
clean_movies_kg	Cleans KG dataset	Section 4.2 Knowledge Graph Querying	Dataset
generate_prompts	Generates prompts	Section 4.3 Prompt Engineering	Prompts
probe_llms	Probes LLMs with prompts	Section 4.4 Large Language Model Probing	Predictions
stats_eval	Statistically analyses the results	Section 5 Results	Results/Summaries Results/Error Matrices Results/Genre Counts Results/Prediction Counts
stats_eval_intermediate	Statistically analyses the intermediate results	Section 4.3 Prompt Engineering	Results/Intermediate
t_tests	Runs statistical significance tests	Section 5 Results	Results/T-Tests
graphs	Generates graphs used throughout this paper	Section 5 Results	Graphs

Table 4: Python files with their descriptions and output folders.

[31] `https://grouplens.org/datasets/movielens/`

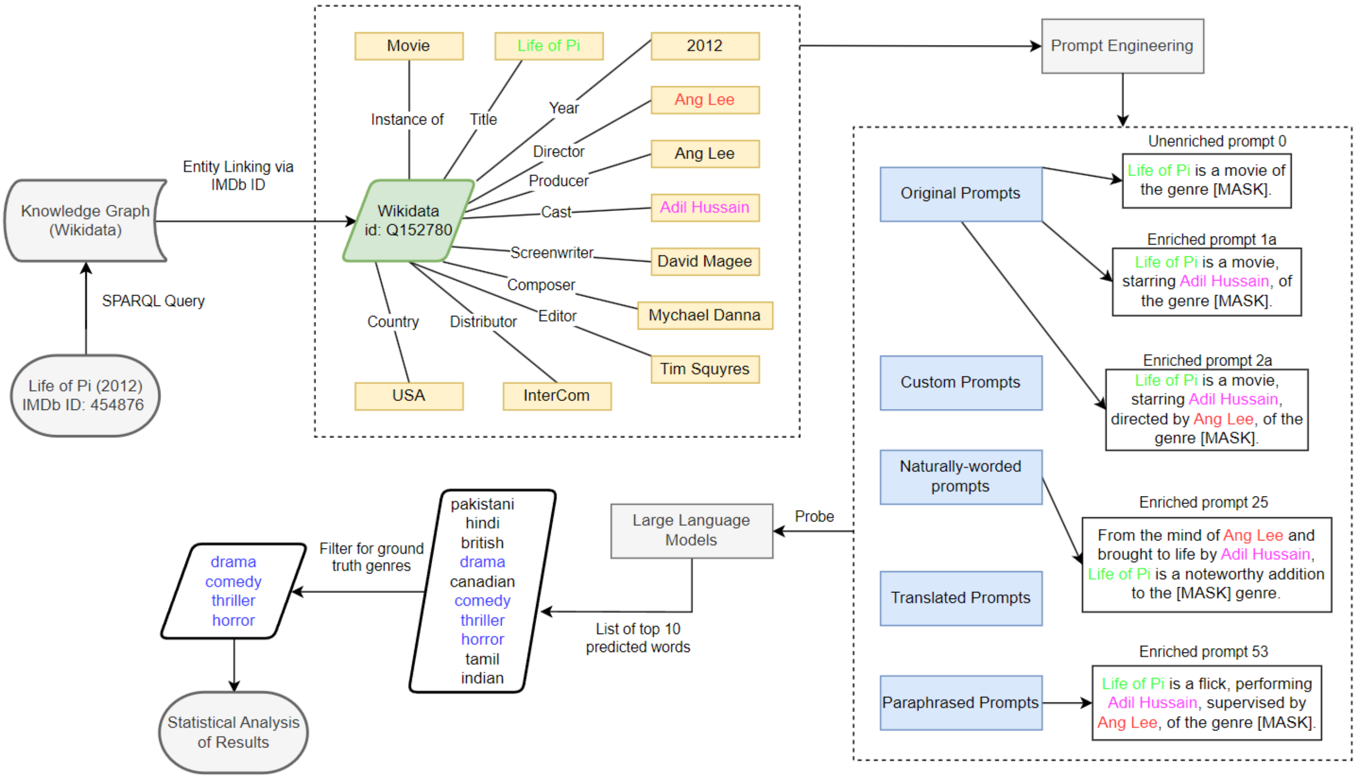

Figure 4: This study's proposed methodology. The movie "Life of Pi" (2012) is used as an example. A subset of prompt examples used in this study are also provided. All prompt types and their explanations can be seen in Table 11.

4.2 Knowledge Graph Querying

Aligned with one of the paper's objectives to validate and scrutinize the findings of Brate et al. [1], it was deemed appropriate to utilize the same properties as those enlisted in the original paper. These properties are enumerated in Table 6. As discussed in Section 3.3.1 Knowledge Graph Selection, Wikidata provides an API[32] that enables users to query Wikidata using SPARQL queries. Each movie was subjected to a SPARQL query[33], illustrated in Figure 6 in the Appendix, to retrieve the corresponding KG properties from Wikidata using the movie's IMDB and TMDB identifiers. Where there are multiple KG values, for example, multiple cast members, the query only retrieves the first property in the set, following Brate et al.'s [1] methodology.

A SPARQL query was executed for each movie, where the Wikidata server returned any requested KG properties that were in English. Movies that lacked any of the KG properties were subsequently removed. Following this filtering process, 8,812 movies with complete KG properties remained in the dataset. The genre distribution of this refined dataset is presented in Table 5.

A pronounced imbalance is evident in the genre distribution; the drama genre, for instance, is overrepresented by a factor of over 250 compared to the animation genre. Such a skewed representation likely does not reflect the broader landscape of American/English cinema. It suggests a selection bias favouring films of certain popular genres, as they might be more comprehensively catalogued in Wikidata. Consequently, some LLMs might display biased genre predictions based on the skewed distribution in their training datasets.

[32] `https://query.wikidata.org/sparql`

[33] The "SPARQLWrapper" Python library[34] was employed to make SPARQL query API calls to Wikidata.

Genre	Count	Percent of Movies
Drama	4959	56.3
Comedy	2804	31.8
Romance	1795	20.4
Thriller	1690	19.2
Action	1376	15.6
Crime	1118	12.7
Adventure	913	10.4
Horror	681	7.7
Mystery	577	6.5
War	517	5.9
Fantasy	420	4.8
Western	373	4.2
Musical	371	4.2
Children	232	2.6
Noir	164	1.9
IMAX	79	0.9
Documentary	63	0.7
Animation	19	0.2

Table 5: Genre distribution for movies retaining complete genre and KG property data, with subsequent percentage calculations. This analysis is based on the refined ML-25 dataset [5] after the exclusion of movies with incomplete KG properties.

WikiData Property	Property Label	Enrichment Text
wdt:P161	cast member	starring
wdt:P57	director	directed by
wdt:P162	producer	produced by
wdt:P58	screenwriter	screenwriter/written by
wdt:P86	composer	music by
wdt:P1040	film editor	edited by
wdt:P577	year	released
wdt:P750	distributed by	distributed by
wdt:P495	country of origin	originating from

Table 6: Movie WikiData Properties used to construct the enriched prompts. "WikiData Property" is the property used in the SPARQL query. "Enrichment Text" is the text used in each of the prompts to express the prompt in a natural style. This table is taken from Brate et al. [1].

4.3 Prompt Engineering

This section describes the construction of the prompts used in this study. Table 11 at the start of Section 5 Results displays each of the prompts and their explanations. To construct the initial prompts, the prompt styles delineated by Brate et al. [1] were adopted. The unenriched prompt, designated as prompt style 0 throughout the paper, was formulated as "TITLE is a movie of the genre [MASK]". Each of the subsequent prompts were then constructed by successively incorporating Wikidata properties, each separated by a comma, resulting in the creation of nine enriched prompt styles, termed 1a-9a.

Some LLMs exhibited significant sensitivity to commas, both in this study and in Brate et al. [1]. Thus, another suite of prompt styles was generated in an identical manner, except for the usage of "and" instead of commas to separate the properties, labelled as 1b-9b. Table 7 illustrates an example of a movie with prompt styles 0 and 1a-24a, while an example of styles 1b-24b can be seen in Table 19 in the Appendix. Throughout the paper, prompts 1a-9a and 1b-9b are referred to as the original prompts.

Prompt	Description
0	Life of Pi is a movie of the genre [MASK].
1a	Life of Pi is a movie, starring Adil Hussain, of the genre [MASK].
2a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, of the genre [MASK].
3a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, of the genre [MASK].
4a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, of the genre [MASK].
5a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, music by Mychael Danna, of the genre [MASK].
6a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, music by Mychael Danna, edited by Tim Squyres, of the genre [MASK].
7a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, music by Mychael Danna, edited by Tim Squyres, released in 2012, of the genre [MASK].
8a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, music by Mychael Danna, edited by Tim Squyres, released in 2012, distributed by InterCom, of the genre [MASK].
9a	Life of Pi is a movie, starring Adil Hussain, directed by Ang Lee, produced by Ang Lee, screenwriter David Magee, music by Mychael Danna, edited by Tim Squyres, released in 2012, distributed by InterCom, originating from United States of America, of the genre [MASK].
10a	The movie Life of Pi starring Adil Hussain, of the genre [MASK].
11a	The movie Life of Pi directed by Ang Lee, of the genre [MASK].
12a	The movie Life of Pi released in 2012, of the genre [MASK].
13a	The movie Life of Pi originating from United States of America, of the genre [MASK].
14a	The movie Life of Pi starring Adil Hussain, directed by Ang Lee, of the genre [MASK].
15a	The movie Life of Pi starring Adil Hussain, released in 2012, of the genre [MASK].
16a	The movie Life of Pi starring Adil Hussain, originating from United States of America, of the genre [MASK].
17a	The movie Life of Pi directed by Ang Lee, released in 2012, of the genre [MASK].
18a	The movie Life of Pi directed by Ang Lee, originating from United States of America, of the genre [MASK].
19a	The movie Life of Pi released in 2012, originating from United States of America, of the genre [MASK].
20a	The movie Life of Pi starring Adil Hussain, directed by Ang Lee, released in 2012, of the genre [MASK].
21a	The movie Life of Pi starring Adil Hussain, directed by Ang Lee, originating from United States of America, of the genre [MASK].
22a	The movie Life of Pi starring Adil Hussain, released in 2012, originating from United States of America, of the genre [MASK].
23a	The movie Life of Pi directed by Ang Lee, released in 2012, originating from United States of America, of the genre [MASK].
24a	The movie Life of Pi starring Adil Hussain, directed by Ang Lee, released in 2012, originating from United States of America, of the genre [MASK].

Table 7: A list of all of the prompt styles 0 and 1a-24a used in this paper, utilizing the movie "Life of Pi" (2012) for illustrative purposes. Successive KG properties introduced, in contrast to the preceding row, are emphasized in red.

Brate et al. [1] adopted a particular style for the 'b' prompts where they integrated "and" before the first KG property, forming prompts akin to "TITLE is a movie **and** starring CAST and directed by DIRECTOR...". This study refined the approach by omitting the initial "and" (highlighted in bold) for grammatical refinement and potential performance enhancement.

Post-processing of the original prompts by the LLMs, an in-depth evaluation was conducted, as described in Section 5 Results. We also established intermediate prompt styles to gauge the efficacy of singular KG properties, with exemplifications in Tables 9 (and 20 in the Appendix). Results in Figures 7, 8, and 9 in the Appendix reveal the pivotal role of the cast, director, year, and country of origin properties in genre predictions, independent of the separation techniques 'a' and 'b'. Subsequently, these KG properties were used to construct a further set of prompt styles. Prompts 10a-24a layer the best-performing KG properties in all possible combinations, separated by commas. Prompt styles 10b-24b replicate this structure, however, separating the KG properties with the word "and" instead. Prompts 10-24 are referred to as the custom prompts throughout this paper.

As documented in Section 2.3 Related Work, both Denny et al. [63] and Ruis et al [65] found that naturally (re)wording their prompts lead to better performance across some LLMs at their respective tasks. As a result, prompts 25-36 were crafted in more natural English compared to the original styles, which primarily just listed KG properties. These styles attempted to embed the mask token somewhere in the middle of the prompts, ensuring the word preceding or following the mask token was 'genre' in the majority of the prompts. The aspiration here was that employing naturally-flowing English, which more closely resembled the text on which some of the LLMs were trained, will enable the LLMs to predict movie genres with greater accuracy. Examples of these prompts are displayed in Table 8. Prompts 25-36 are referred to as the naturally-worded prompts throughout this paper.

Prompt	Description
25	From the mind of Ang Lee and brought to life by Adil Hussain, Life of Pi is a noteworthy addition to the [MASK] genre.
26	With Life of Pi, Ang Lee brings a new twist to the [MASK] genre, featuring powerful performances by Adil Hussain.
27	The [MASK] genre is beautifully represented in United States of America through the movie Life of Pi, featuring the unique performance of Adil Hussain.
28	Through the lens of Ang Lee, Life of Pi blends gripping performances by Adil Hussain with the nuanced themes of the [MASK] genre.
29	Life of Pi is a remarkable exploration of the [MASK] genre, driven by the stellar direction of Ang Lee and compelling acting from Adil Hussain.
30	Immersing audiences in the [MASK] genre, Ang Lee creates a cinematic gem with Life of Pi, featuring a standout performance by Adil Hussain.
31	A film released in 2012 from United States of America, Life of Pi features Adil Hussain and falls into the [MASK] genre under the direction of Ang Lee.
32	Life of Pi, a masterpiece in the [MASK] genre from 2012, reflects Ang Lee's vision and United States of America's culture, starring Adil Hussain.
33	Ang Lee crafts a vibrant narrative within the [MASK] genre in 2012's Life of Pi, encapsulating the heartbeat of United States of America with an unforgettable performance by Adil Hussain.
34	Life of Pi, a cinematic treat from United States of America released in 2012, weaves a compelling [MASK] narrative under the mastery of Ang Lee, featuring Adil Hussain.
35	Under the masterful direction of Ang Lee, Life of Pi was released in 2012, representing the unique spirit of United States of America's film industry, while also creating a fresh narrative in the [MASK] genre, featuring the remarkable talents of Adil Hussain.
36	In 2012, the film world was enriched by Life of Pi, a significant [MASK] genre movie hailing from United States of America, guided by the innovative vision of director Ang Lee and showcasing the notable performances of Adil Hussain.

Table 8: A list of all of the naturally-worded prompt styles 25-36 used in this paper, utilizing the movie "Life of Pi" (2012) for illustrative purposes. Successive KG properties introduced, in contrast to the preceding row, are emphasized in red.

KG Property	Description
Cast	Life of Pi is a movie starring Adil Hussain, of the genre [MASK].
Director	Life of Pi is a movie directed by Ang Lee, of the genre [MASK].
Producer	Life of Pi is a movie produced by Ang Lee, of the genre [MASK].
Screenwriter	Life of Pi is a movie screenwriter David Magee, of the genre [MASK].
Composer	Life of Pi is a movie music by Mychael Danna, of the genre [MASK].
Editor	Life of Pi is a movie edited by Tim Squyres, of the genre [MASK].
Year	Life of Pi is a movie released 2012, of the genre [MASK].
Distributor	Life of Pi is a movie distributed by InterCom, of the genre [MASK].
Country	Life of Pi is a movie originating from United States of America, of the genre [MASK].

Table 9: A list of all of the intermediate 'a' prompt styles used in this paper, utilizing the movie "Life of Pi" (2012) for illustrative purposes. KG properties and their labels are emphasized in red.

Building upon the insights from the initial custom, original, and naturally-phrased prompt results, we expanded our investigation into sophisticated prompt generation methodologies, as discussed in Section 2.3 Related Work. This extension utilized the top-tier prompt styles, notably styles 2a, 7b, 9b, 31, and 32.

One technique of interest was the round-trip translation, a method explored in Jiang et al. [22]. Herein, the aforementioned best-performing prompts underwent translation to a foreign language and subsequently back-translated to English, producing a set of 10 novel prompts. French and German were the selected intermediary languages, primarily due to their recurrent utilization in similar contexts [22][23][24]. The Helsinki MarianMT LLM, discussed in Section 2.2.6 Other Large Language Models, executed these translations. Mallinson et al. [23] assert that using a Neural Machine Translation approach, as employed in this study, ensures holistic consideration of the sentence during translation, emphasizing the retention of semantic integrity, which ultimately leads to more accurate translations. Prompts 37-46 are referred to as the translated prompts throughout the paper.

Various paraphrasing techniques were discussed in Section 2.3 Related Work, with emphasis being placed on the retention of semantic integrity during the paraphrasing. Another way of performing this paraphrasing while still keeping semantic integrity intact is by using an LLM fine-tuned for paraphrasing tasks. As previously mentioned, this study makes use of FLAN-T5 [57] for this task, which has been fine-tuned for rephrasing tasks, as well as a range of other NLP tasks. Each of the aforementioned best-performing prompts were paraphrased, generating 5 new prompts. Prompts 47-51 are referred to as the T5-paraphrased prompts throughout the paper[35].

Another technique used in this study is the thesaurus paraphrasing technique taken from Yuan et al. [19], where we took the best-performing prompts for all of the LLMs and manually rewrote the prompts, replacing the connecting words and phrases (between the KG properties) with their thesaurus alternatives. We use the same thesaurus website WordHippo[36] as Yuan et al. [19]. For each term or phrase, we discerned the most pertinent definition and subsequently replaced it with its first three synonymous alternatives. This technique resulted in 3 new prompts for each of the best-performing prompts, labelled 52-66, examples of which can be seen in Table 21 in the Appendix. Prompts 52-66 are referred to as the thesaurus-paraphrased prompts throughout the paper.

To circumvent potential translation or rephrasing of the mask token in both the round-trip translation and the T5-paraphrasing approaches, we utilized specific placeholder tokens. A string of punctuation was identified as optimal for MarianMT, whereas FLAN-T5 [57] exhibited a preference for the placeholder "comedy". Instances where the placeholder was erroneously removed were excluded from LLM probing, and subsequently omitted from average recall score computations. Retrofitting the mask tokens post-translation would likely jeopardize the semantic integrity of the prompts.

[35]Examples of the round-trip translated and T5-paraphrased prompts are not presented in the report as they are all different for different movies, but can be found in the supplemental files.

[36]`https://www.wordhippo.com/`

In this study, potential techniques for automatic prompt generation as described in Haviv et al. [25] and Zhou et al. [26] were assessed. These techniques involve an iterative process to generate, evaluate, and refine prompts to achieve an optimal structure within defined constraints. However, it is essential to note that such techniques can be interpreted as a form of pre-training due to their iterative nature. Given that our primary interest lies in genuine zero-shot learning, the implementation of such methodologies, which diverge from the strict zero-shot paradigm, was dismissed.

Regarding the prompt styles, while the paper elucidates styles 1a-24a and 1b-24b, the code incorporates additional variations, namely 1c-24c and 1d-24d. These latter styles mirror 1a-24a and 1b-24b, except for the replacement of the string "[MASK]" with "<mask>", catering to LLMs with variant mask token preferences[37]. To streamline communication and maintain clarity, following the LLM data processing phase, prompt styles 1c-24c and 1d-24d were renamed to 1a-24a and 1b-24b respectively; prompts 0a and 0c were renamed to 0; and prompts 25a-66a were renamed to 25-66 in the "Recall" and "Results" folder's CSV files.

4.4 Large Language Model Probing

As mentioned in Section 3.3.2 Large Language Model Selection, Hugging Face[38] provides a vast array of LLMs that can be downloaded and utilized. The python transformers library[39] offers APIs to download these pre-trained models from Hugging Face and was utilized in this paper's implementation to download the LLMs. Furthermore, the Python torch library[40] has been employed to provide GPU acceleration when processing data, which significantly enhances runtime efficiency compared to relying solely on a system's CPU. Brate et al. [1] made use of the Hugging Face pipeline API[41], while this study's implementation manually handled several aspects of the model prediction process, including loading each specific model's tokenizers, generating input tensors, applying the model, and interpreting the output. The reason for this manual handling is that using the original pipeline method would not generate a list of the top 10 predictions necessary for recall@10, as the pipeline API does not offer a way to observe an exhaustive list of the LLM predictions.

Upon downloading the respective LLMs, each prompt was sequentially processed. A comprehensive list of unique genres in the dataset, representing the ground truth genres, was compiled. The top 10 predictions from each LLM, which were free of any special characters and whitespaces, were subsequently saved. This approach, and the subsequent computation of recall scores, is elaborated upon in Section 5 Results.

After manually analysing the results, a number of tweaks were made to the predicted list of words before calculating recall scores. Within the topmost 100 words predicted by each LLM, any terminology bearing significant similarity to an established genre in terms of semantic content was swapped with the latter. This nuanced step was necessary to address the disproportionately low recall scores observed in some LLMs, even when they effectively predicted near-synonyms of the genres. Table 10 displays all substituted terms, with Table 16 in Section 5.2.2 Exploring Divergences in Performance showing the number of swaps made. It is imperative to note that only terms with closely aligned meanings underwent substitution. For instance, the term "dramatic" was not interchanged with "drama", as its semantic inclination might resonate more with the "action" genre than the "drama" genre. Stemming and lemmatization methods were considered, but were deemed inappropriate, as, for example, the lemma of the words "romantic" and "romance" is not the same (both words remain the same), and the stem may also not be the same ("romant" and "romanc" respectively) depending on if the Porter, Snowball or other methods are used.

After processing all of the prompts, all of the prediction CSV files were merged, such that the results for all 91 prompt styles were stored in one CSV file per LLM, facilitating the subsequent statistical analysis. The 18 intermediate prompts are stored in a separate folder. In the next chapter, we will discuss the results of the methodology described in this chapter.

[37] Table 22 in the Appendix provides a breakdown of which LLMs make use of which mask styles within the code.
[38] `https://huggingface.co/models?pipeline_tag=fill-mask&language=en&sort=downloads`
[39] `https://huggingface.co/docs/transformers/index`
[40] `https://pypi.org/project/torch/`
[41] `https://huggingface.co/docs/transformers/main_classes/pipelines`

Originally Predicted Word	**Replaced Word (Genre)**
Romantic	Romance
Love	Romance
Music	Musical
Comedic	Comedy
Comedies	Comedy
Animated	Animation

Table 10: Pairs of swapped predicted words synonymous to the ML-25 [5] dataset's ground truth genres.

5 Results

The heatmap Figures 10-15 in the Appendix, illustrate the average recall scores for each LLM, prompt style, and recall@n (R@n). For the recall@1 (R@1), a score of 1 would denote the perfect accuracy of an LLM's first prediction, while recall@5 (R@5) and recall@10 (R@10) represent the average accuracy of the LLM's initial 5 and 10 predictions respectively.

For a concrete example, consider the movie "GoldenEye" (1995), classified under action, adventure, and thriller genres. BERT's predictions with prompt style 0 for this movie were: thriller, comedy, noir, genre, cinema, film, horror, drama, adventure, and trilogy. Evaluating R@5, the first 5 words are taken, among which 3 are valid genre terms (thriller, comedy, noir). Out of these, thriller was accurately predicted, leading to an R@5 score of $\frac{1}{3} = 0.33\overline{3}$. R@10 evaluation proceeds similarly, considering the first 10 predictions. Scores of 0 are assigned if none of the initial n values match the ground truth genres at R@n.

The highest accuracy prompt style for each LLM is highlighted in red in each of the heatmaps, with the best-performing prompt style across all prompt styles displayed in Table 12. As previously mentioned, all of the prompt styles and their explanations are available in Table 11. In this chapter, any LLMs labelled "x large (v2)" will be referred to as "x", and the words "prompt" and "style" are used interchangeably.

Initially, the most salient findings of this paper are presented. Subsequently, these results are critically examined and contextualized in relation to the underlying architectures and training datasets for each LLM. Furthermore, a comprehensive error analysis is presented. Finally, a comparative assessment with Brate et al. [1] is also undertaken.

Prompt Styles	Referred to as	Explanation	Examples
0	Unenriched original prompts	No KG property information following Brate et al. [1]	Table 7
1a - 9a 1b - 9b	Enriched original prompts	KG properties sequentially added. Separated either by (a) commas or (b) the word "and", following Brate et al. [1]	Tables 7 and 19
10a - 24a 10b - 24b	Enriched custom prompts	Constructed using the most accurate KG properties in exhaustive combinations - cast, director, year, country. Separated either by (a) commas or (b) the word "and".	Tables 7 and 19
25 - 36	Naturally-worded prompts	Naturally-worded prompts using the most accurate KG properties - cast, director, year, country.	Table 8
37, 38 (2a) 39, 40 (7b) 41, 42 (9b) 43, 44 (31) 45, 46 (32)	Translated prompts	Best-performing prompts 2a, 7b, 9b, 31, and 32 round-trip translated to either French (odd numbers) or German (even numbers) and back to English using the MarianMT LLM [56].	Supplemental files
47 (2a) 48 (7b) 49 (9b) 50 (31) 51 (32)	T5-Paraphrased prompts	Best-performing prompts 2a, 7b, 9b, 31, and 32 paraphrased using the FLAN-T5 LLM [57].	Supplemental files
52, 53, 54 (2a) 55, 56, 57 (7b) 58, 59, 60 (9b) 61, 62, 63 (31) 64, 65, 66 (32)	Thesaurus-paraphrased prompts	Best-performing prompts 2a, 7b, 9b, 31, and 32 paraphrased using the thesaurus method based on Yuan et al. [19].	Table 21

Table 11: Prompt styles used in this study explained, along with what they are referred to throughout this paper.

LLM	R@n	Best Prompt	Mean Difference (3 SF)	Test Statistic (3 SF)	p-values
BERT	1	50	0.365	-55.2	0*
	5	50	0.304	-59.9	0*
	10	50	0.296	-60.7	0*
RoBERTa large	1	50	0.334	-50.1	0*
	5	9b	0.373	-89.6	0*
	10	9b	0.303	-76.0	0*
BART large	1	50	0.458	-79.3	0*
	5	50	0.586	-128	0*
	10	50	0.683	-159	0*
ALBERT large v2	1	50	0.428	-75.4	0*
	5	50	0.604	-145	0*
	10	50	0.735	-200	0*

Table 12: Best performing prompt style for each LLM and R@n. The mean difference is the average recall score difference between the unenriched prompt (0) and the given prompt style. One-tailed, directional, dependent t-tests have been performed. Full t-test results for all prompt styles are available in the supplemental files. *Note: any p-value smaller than 5×10^{-324} has been rounded down to 0 in Python.

LLM	R@n	Brate et al. [1] Best Prompt	This Studies Best Prompt	Mean Difference (3 SF)	Test Statistic (3 SF)	p-values
BERT	1	2b	50	0.222	-30.00	2.65×10^{-187}
	5	2a	50	0.093	-20.87	2.12×10^{-94}
RoBERTa large	1	9b	50	0.099	-14.38	1.40×10^{-46}
	5	9b	9b	0	-	-

Table 13: Best-performing prompt style for each LLM and R@n in both Brate et al. [1] and this study. The mean difference is the average recall score difference between the best-performing prompt from Brate et al. [1] and this study's best-performing prompt. One-tailed, directional, dependent t-tests have been performed. Full t-test results for all prompt styles are available in the supplemental files.

5.1 Performance Evaluation of Large Language Models

The performance of each LLM concerning the distinct prompt styles is deliberated in this section. For the highest-performing prompt style for each LLM and R@n, we evaluated the statistical significance of this style against the unenriched prompt 0 using a one-tailed, directional, dependent t-test, as depicted in Table 12. The mean difference denotes the average difference in recall scores between the unenriched prompt style 0 and the best style[42]. The null hypothesis is that the mean difference is 0, and the alternative hypothesis is that the mean difference is greater than 0. At a significance level of 0.01, the results indicate that all LLMs significantly predict genres with greater accuracy using the enriched prompts compared to the unenriched prompt 0.

For a subset of the LLMs and R@n at a significance level of 0.01, we also demonstrated that the prompts developed in this paper yield superior performance compared to the original prompts from Brate et al. [1] by also performing a one-tailed, directional, dependent t-test, as visible in Table 13. The mean difference denotes the average difference in recall scores between the best-performing prompts in Brate et al. [1] (with the data produced in this study for their best prompt styles being used) and the best prompts found in this study.

[42] Mean difference = Mean score (Best prompt) - Mean score (prompt 0)

5.1.1 BERT

In general, BERT achieved a higher accuracy with the original prompts compared to the custom and naturally-worded prompts. At R@1 and R@5, BERT's best original/custom/naturally-worded prompt performance was with style 2a, although it is noted that styles 1a, 1b, and 2b were always quite close behind, suggesting that BERT's optimal performance was triggered by minimal KG information. Specifically, the most effective original prompts for these R@n incorporated an actor and the director. This observation is consistent with Figures 7, 8, and 9 in the Appendix, which indicate that cast and director are the best-performing KG properties for BERT.

For the custom prompts 10-24, BERT's performance was markedly low at R@1, implying a pronounced unsuitability for predicting a singular genre. However, performance improved at higher R@n, albeit not matching the levels achieved with original prompts. This suggests that the introductory phrase "TITLE is a movie..." is more conducive to BERT's understanding than "The movie TITLE is...". Recall scores for styles 10-24 displayed considerable variability, with ranges of 0.004-0.147 at R@5 and 0.041-0.387 at R@10. This underscores the impact of specific KG properties on BERT's efficacy.

Regarding the naturally-worded prompts 25-36, BERT showcased an enhanced performance. However, the recall score ranges for prompts 25-36 remain expansive: 0.077-0.229, 0.109-0.476, and 0.177-0.735 for R@1, R@5, and R@10 respectively. Among these, styles 27, 34, and 36 underperformed, while style 32 consistently excelled. These variations in recall scores demonstrate BERT's heightened sensitivity to prompt wording.

In analyzing the comparative performance between prompt pairs 1a-24a and 1b-24b, BERT's results appear nuanced. Notably, for the majority of the initial styles 1-9, BERT demonstrated superior performance with the 'b' styles, as demonstrated by the predominantly darker hues in Figures 10, 11, and 12. However, style 2 represents an outlier, where 2a consistently outperforms 2b across all R@n metrics. Moreover, it is remarkable that 2a is the highest performer among original prompts at R@1 and R@5. Conversely, for prompts 10-24, BERT showed a preference for 'a' styles across all R@n measurements.

When broadening the scope to encompass all prompt styles, BERT's pinnacle of performance was observed with T5-paraphrased prompts 47-51, with the best-performing style being prompt 50 (31 T5-paraphrased). Intriguingly, the naturally-worded prompt where BERT excelled was style 32, not 31, indicating a disparity in performance when transformed into T5-paraphrased styles 51 versus 50.

With respect to the translated prompts 37-46, the visual representation in Figures 13, 14, and 15 suggests a predominant favorability towards the French translations (odd-numbered) as opposed to their German counterparts. An anomaly, however, emerges with prompts 43 and 44 (31 translated), where BERT, in tandem with RoBERTa and ALBERT, displayed superior performance with the German variant.

Finally, the thesaurus-paraphrased prompts, spanning 52-66, presented variable outcomes. For instance, prompt 56 (7b thesaurus-paraphrased) aligns in performance with other styles, whereas prompt 60 (9b thesaurus-paraphrased) exhibited notably poor performance across all R@n measures. What renders these results particularly enigmatic is the fact that while BERT was highly compatible with prompt 2a among original/custom prompts, its thesaurus-paraphrased versions (52-54) did not maintain this superiority, hinting at the paramount importance of synonym selection over sentence structure in the paraphrasing process.

5.1.2 RoBERTa Large

In comparative evaluations, RoBERTa consistently surpassed BERT in terms of R@n performance across the majority of the original prompts. Notably, RoBERTa demonstrated optimal performance with original styles 7b, 9b, and 9b for R@1, R@5, and R@10, respectively. It is pertinent to highlight that these styles prominently feature the year and country as the terminal KG properties preceding the mask token. This observation aligns with the data presented in Figures 7, 8, and 9 in the Appendix, which underscore the significance of the year and country KG properties in augmenting RoBERTa's performance at all R@n metrics.

The alternating patterns observed in the pairs of original prompts 1a-24a and 1b-24b, as illustrated in Figures 10, 11, and 12, accentuate RoBERTa's enhanced compatibility with the 'b' styles. Nevertheless, it is imperative to acknowledge that the recall differentials for styles 7-9 appear relatively condensed when compared against styles 3-6. A recurrent theme with the 'b' styles is the positive correlation between an increase in KG properties in the prompt and improved recall metrics.

Analogous to BERT's performance metrics, RoBERTa exhibited subpar recall scores when subjected to specific original prompts, specifically 4a and 6a across all R@n. These prompts prioritize the inclusion of the screenwriter and editor KG properties. Figures 7, 8, and 9 corroborate this observation, pinpointing screenwriter and editor KG properties as potential bottlenecks in RoBERTa's performance.

In analyzing the results related to custom prompts, RoBERTa exhibited a substantial variability. Specifically, for styles 10-24, there was a strikingly diverse range of outcomes, with R@10 values fluctuating between 0.104 and 0.772. It was observed that RoBERTa demonstrated a consistent preference towards the 'b' styles, with the sole exception being styles 12-14. Notably, style 14b consistently underperformed for all custom prompts.

Regarding the naturally-worded prompts 25-36, RoBERTa's recall scores varied, though none of the styles in this category plummeted to the levels seen in the 10-24 range. Among them, styles 31 and 32 consistently registered the highest recall scores.

When considering all of the prompts, RoBERTa performed best with style 50 (31 T5-paraphrased) at R@1, with style 9b leading to the best performance for the other R@n. Styles 39 and 40, which were style 7b round-trip translated to both French and German, narrowly missed outperforming style 9b across all R@n metrics. A mere 0.001 point in recall scores separated styles 9b and 39 at R@5. A noteworthy observation is the dichotomous performance of RoBERTa on round-trip translated prompts: styles 39-42 ranked among the top, while styles 38, 43, and 45 languished at the bottom. Contrary to BERT, RoBERTa did not demonstrate a discernible preference between French and German.

RoBERTa's response to thesaurus-paraphrased prompts 52-66 varied significantly. A discernible trend was that if RoBERTa excelled with the primary prompt, it similarly performed well with its thesaurus-paraphrased counterparts. This observation is substantiated by Figures 13, 14, and 15. However, style 63 was an outlier due to its selection of synonyms. Table 21 suggests that the terms "government" and "resorts" in prompt 63 may have confounded RoBERTa's performance.

5.1.3 BART Large

Figure 10 illustrates BART's suboptimal R@1 performance, demonstrating marginal performance enhancements from the inclusion of additional KG properties. However, Figures 11 and 12 indicate a progressive increase in BART's recall performance at R@5 and R@10, respectively, with prompt 2a outperforming others. In comparison to BERT and RoBERTa, BART's performance remains inferior for the majority of the original prompts. Notably, while KG property enrichment improves BART's performance, there is a discernible decline beyond style 2a, with style 7a being particularly underwhelming. This coincides with the Appendix Figures 8 and 9, which emphasize the significance of the director as an influential KG property for BART.

Table 12 accentuates that prompt 50 (31 T5-paraphrased) registered the highest recall scores across all R@n. The prominence of this prompt is further solidified by Figures 13, 14, and 15, which illustrate its superior performance in relation to other prompts.

BART's performance was much better with the naturally-worded prompts 25-36 compared to the custom prompts 10-24. At R@1, Figure 13 shows us that prompts 25-36 also perform better than the original prompts, although at R@5 and R@10, Figures 14 and 15 show us that the gap in performance drops markedly. For prompts 25-36, prompt 31 was indeed the best-performing prompt at all R@n by a decent margin, so it makes sense that BART's best-performing prompt (50) is a T5-paraphrased version of prompt 31.

Figures 11 and 12 elucidate the oscillatory nature of BART's performance between the paired prompts 'a' and 'b'. Predominantly, BART exhibits superior performance with the 'b' styles, for both original and custom prompts. This observation is underscored by the alternating patterns at R@5 and R@10, which predominantly align with the 'b' styles or present a recall score marginally above 0. Intriguingly, both BERT and BART display a common trait: despite the general preference for 'b' styles, for style 2, BART is distinctly better aligned with 2a, mirroring the pattern seen in BERT's optimal performance with the original prompt. In scenarios restricted to a singular KG property, as depicted in Figures 8 and 9, BART's predilection for comma usage is evident over the conjunction "and".

Contrastingly, when exposed to thesaurus-paraphrased prompts, BART's prowess falters in comparison to BERT and RoBERTa. Even though styles 2a, 31, and 32 manifest commendable performance for BART, its efficacy diminishes for their thesaurus-paraphrased counterparts, suggesting BART's optimal performance is achieved with succinct and direct language.

5.1.4 ALBERT Large v2

ALBERT's performance stands out due to distinct differences observed when utilizing original, custom, and other prompt variations. As depicted in Figures 11 and 12 for R@5 and R@10 respectively, ALBERT demonstrates inferior results among all the LLMs for both original and custom prompt styles, although it is noted that for R@1 as illustrated in Figure 10, certain prompts yield outcomes that are marginally better than BART. In fact, ALBERT's highest recall scores with the original prompts are 0.008, 0.014, and 0.034 for each R@1, R@5 and R@10 respectively.

Interestingly, when employing the naturally-worded prompts (25-36), ALBERT's efficacy increases substantially, surpassing both BERT and BART for specific styles, and even outperforming RoBERTa with prompts 26 and 29 at select R@n values. For R@1, akin to BERT, prompt 32 emerged as the most effective, while for R@5 and R@10, prompts 26 and 31 exhibited superior results.

Remarkably, prompt 50 (31 T5-paraphrased) consistently outperformed other prompts across all R@n measures. With this prompt, ALBERT achieved an R@1 recall score of 0.430. The subsequent best-performing style, 49 (9b T5-paraphrased), lagged significantly with a recall score of 0.241, reflecting a substantial gap of 0.189. It's notable that while prompt 9b only yielded a score of 0.010 at R@1, its T5-paraphrased counterpart displayed remarkable improvement. Meanwhile, as anticipated, ALBERT's results with most of the thesaurus-paraphrased prompts (52-60) were suboptimal. However, it exhibited commendable performance with prompts 61-66, correlating with its previous affinity for prompts 31 and 32. It is noted that the variability between the recall scores 61-63 is quite staggering, where, just like for RoBERTa and BART, prompt 63 threw ALBERT off with its nuanced language.

5.2 Discussion

In this section, we discuss the rationale behind the findings presented in Section 5.1 Performance Evaluation of Large Language Models, using the information about the underlying architectures and training datasets of each LLM, as discussed in Section 2.2 Large Language Models. We also compare our results to the results of Brate et al. [1] with the help of Table 15, which displays the best performing 'a' and 'b' original prompt styles in both Brate et al. [1] and this study, along with their respective t-tests. We also discuss the genre distributions (error matrices) and the most commonly predicted words for each LLM.

5.2.1 Prediction Analysis

As previously noted, Table 5 provides the genre counts across the filtered dataset. Error matrices for each LLM at R@1 averaged across all prompt styles can be seen in the Appendix in Tables 25, 26, 27, and 28, with the other R@n error matrices, as well as error matrices for every single prompt style separately at each R@n for each LLM, available in the supplemental files. The values are averaged across all prompt styles and divided by the total number of prompt styles (91) to display the average genre counts for one prompt style for the whole filtered dataset of 8,812 movies[43]. Rows represent the true genres, while columns represent the predicted genres. The diagonal cells display the true positives, while the non-diagonal cells represent the false positives (for the row genre) and false negatives (for the column genre). For example, if the genres of a movie were action and crime, but the LLM predicted drama, the values for both action and crime would be incremented by 1 (before being divided by the total prompt styles) to show that the LLM misclassified both of the movie's genres. As expected, the diagonal cells have higher proportions of true positives when

[43] The error matrices presented in this report are based on raw counts rather than normalized values. The decision to abstain from normalization was made to preserve the direct interpretation of absolute errors for each combination of true and predicted genres. This representation provides a granular view of the discrepancies between the model's predictions and the ground truth.

looking at the best-performing style's error matrices in the supplemental files, compared to the unenriched prompt 0.

For all of the LLMs, the drama, comedy, romance and thriller genres were the most selected genres, which matches the top 4 genres displayed in Table 5. The ML-25 dataset [5] serves as a commendable representation of English movie genre distributions, attributable to its vastness. Nevertheless, potential biases may arise due to the exclusion of movies lacking KG properties. It is plausible that when LLMs encounter movie genres during their training, the genre distribution aligns closely with the comprehensive ML-25 dataset [5]. The error matrices elucidate that LLMs exhibit a pronounced tendency to misclassify horror movies, often categorizing them as drama or comedy. Similarly, comedy movies are frequently misclassified as either drama or romance. Given the overwhelming presence of the drama genre in the dataset, it remains less susceptible to misclassification. Unlike the other LLMs, RoBERTa predicted drama at R@1 the most by quite a large margin, although an extreme amount of it's drama predictions were actually thriller and comedy movies.

Table 14 at R@1 (as well as Tables 23 and 24 in the appendix at the other R@n) display the genre prediction distributions for each LLM, normalized column-wise. We can see that, at R@1:

- BERT selects the comedy and horror genres just under 60% of the time.
- RoBERTa selects the thriller, comedy, and horror genres just under 80% of the time.
- BART selects the horror, drama, and comedy genres just under 70% of the time.
- ALBERT selects the horror and comedy genres just under 80% of the time.

These findings suggest a consistent pattern where LLMs seem inclined to select a common set of 2 or 3 genres approximately 70% of the time, although the specific genres differ inter-LLM. Interestingly, the IMAX genre was universally overlooked by all LLMs, likely due to its infrequency in prevalent training datasets. Beyond the IMAX genre, the children genre is notably underrepresented in BERT, RoBERTa, and ALBERT, while BART refrains from selecting the noir genre entirely.

Genre	**BERT**	**RoBERTa Large**	**BART Large**	**ALBERT Large v2**
Action	0.0025	0.016	0.071	0.00069
Adventure	0.0064	0.0039	0.0026	0.0038
Animation	0.00049	0.0007	0.0022	0.00024
Children	5.1×10^{-5}	2.1×10^{-5}	0.0022	0
Comedy	**0.31**	0.29	0.14	0.3
Crime	0.0024	0.0046	0.017	0.0072
Documentary	0.015	0.012	0.0068	0.058
Drama	0.085	0.027	0.16	0.044
Fantasy	0.019	0.016	0.0061	0.0039
Horror	0.28	0.18	**0.41**	**0.5**
IMAX	0	0	0	0
Musical	0.019	0.01	0.042	0.032
Mystery	0.0006	0.0011	0.0031	0.0017
Noir	0.09	0.00017	0	0.0033
Romance	0.014	0.075	0.078	0.019
Thriller	0.058	**0.32**	0.0024	0.017
War	0.0048	0.0027	0.0024	0.001
Western	0.097	0.042	0.061	0.003

Table 14: LLM genre counts at R@1 (2 SF), normalized column-wise. The most common genres selected per LLM are highlighted in bold.

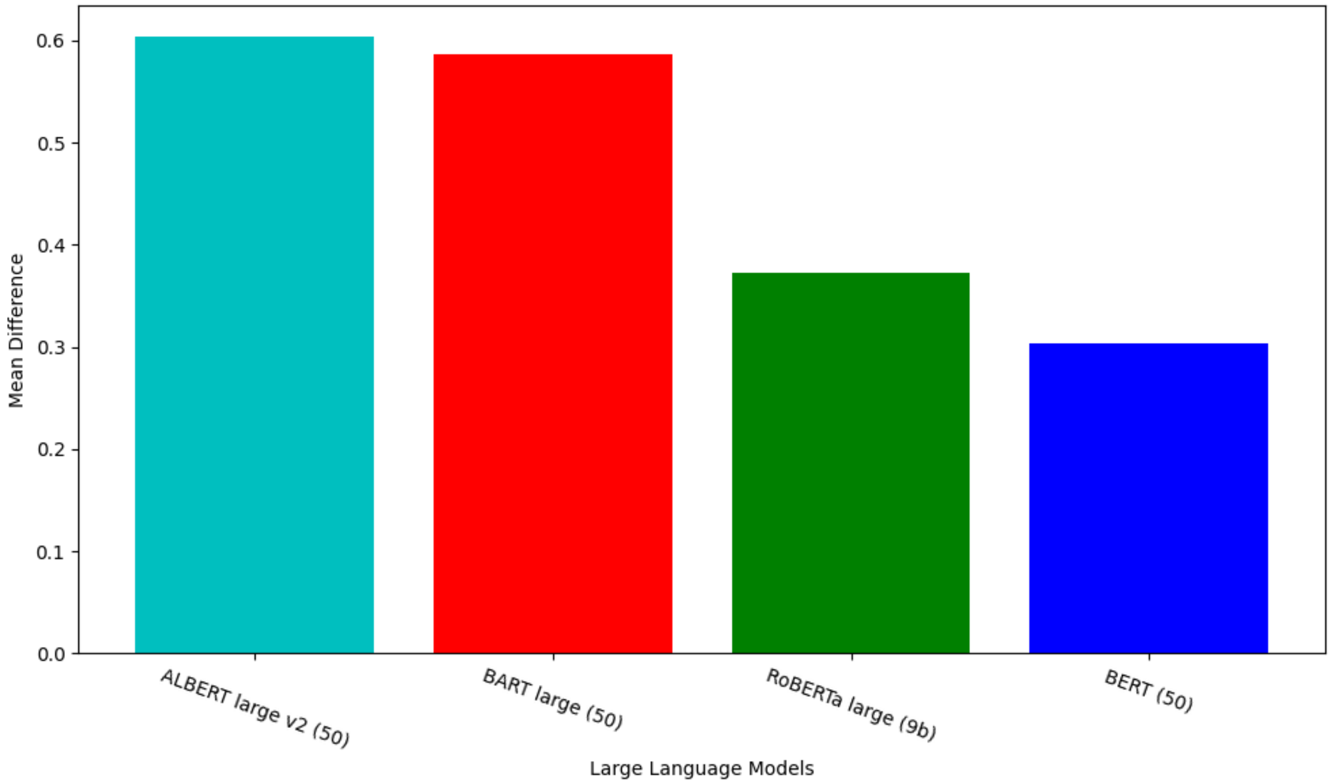

Figure 5: Mean Differences of the best-performing prompts (in brackets) compared to the unenriched prompt 0 for each LLM at R@5.

In the Appendix, Tables 29, 30, 31, and 32 elucidate the top 50 words predicted by each LLM at R@1 prior to any filtration based on the true genres. BERT, RoBERTa, and ALBERT predominantly forecast the term ”genre”. In stark contrast, BART's proclivity is towards function words[44], with ”of”, ”was”, ”and”, and ”is” dominating its R@1 predictions. This pattern in BART's predictions elucidates its underwhelming R@1 performance but accounts for its improvement at higher R@n values, given the expanded predictive range. Notably, while all LLMs occasionally opted for nationalities like ”british” or ”french”, BART exhibited an over-reliance with 15 of its top 50 predictions being nationalities. Several LLMs also projected genres absent from the ML-25 dataset [5], such as ”history/historical”, ”bollywood”, and ”dating”. General cinematic terminologies, namely ”film(s)” and ”trilogy”, were ubiquitously predicted. Curiously, BERT exhibited a penchant for musical genres, evidenced by its predictions like ”pop”, ”jazz”, and ”rock”. Except ALBERT, all LLMs ventured predictions like ”classic” or ”classical”.

In Figure 5 (as well as Figures 16 and 17 in the Appendix), we present the mean differences associated with the most effective prompt styles for each R@n measure. Notably, despite its subpar performance with numerous original and custom prompts, ALBERT - when only evaluating its best-performing prompts - surpasses all other LLMs in performance, followed in order by BART, RoBERTa, and, as expected, BERT.

[44]Function words are those with minimal lexical significance but play pivotal roles in expressing grammatical relationships within sentences.

Study	LLM	R@n	Best Original Prompt	Mean Difference (3 SF)	Test Statistic (3 SF)	p-values
Brate et al. [1]	BERT	1	2a	0.0245	8.33	0*
		5	2a	0.0672	21.0	0*
		1	2b	0.0252	8.47	0*
		5	2b	0.0506	15.2	0*
	RoBERTa Large	1	7a	0.125	43.0	0*
		5	7a	0.358	86.5	0*
		1	9b	0.144	47.7	0*
		5	9b	0.378	92.5	0*
This study	BERT	1	2a	0.180	-37.1	0**
		5	2a	0.211	-53.8	0**
		1	1b	0.170	-33.9	0**
		5	1b	0.200	-51.4	0**
	RoBERTa Large	1	7a	0.174	-36.5	1.16×10^{-272}
		5	7a	0.228	-56.6	0**
		1	9b	0.237	-47.0	0**
		5	9b	0.373	-89.6	0**

Table 15: Best performing 'a' and 'b' original prompt styles for each LLM and R@n in Brate et al. [1]. as well as in this study. The mean difference is the average recall score difference between the unenriched prompt (0) and the given prompt style. One-tailed, directional, dependent t-tests have been performed. Full t-test results for all prompt styles are available in the supplemental files. *Note: Brate et al. [1] rounded their p-values to 3 SF. **Note: any p-value smaller than 5×10^{-324} has been rounded down to 0 in Python.

	BERT	RoBERTa Large	BART Large	ALBERT Large v2	Total
Love	55	124	196	4	379
Romantic	325	442	2606	147	3520
Comedic	3	155	6	123	287
Comedies	389	0	0	12	401
Animated	44	95	32	75	246
Music	768	149	102	164	1183
Total	1584	965	2942	525	6016

Table 16: Each of the genre-synonymous replacement words, as documented in Table 10, divided by the total number of prompts (91) so that the table displays an average number of replacements for a single prompt style across the whole cleaned dataset at R@10.

5.2.2 Exploring Divergences in Performance

In examining BERT's performance in relation to prompt length, it was observed that BERT consistently achieved the highest recall scores with shorter prompts that exhibited minimal KG properties across various R@n: 1, 2, 32, 37 (2a translated), 49 (31 T5-paraphrased), and 50 (32 T5-paraphrased). As elaborated in Section 2.2.2 BERT, while BERT exhibits proficiency in comprehending immediate context before and after a given token, it occasionally struggles with "high-order, long-range dependencies" within sentences [60], which culminates in reduced accuracy with the longer prompts.

Table 15 showcases findings by Brate et al. [1], where prompt styles 2b and 2a emerged as the optimal styles for BERT at R@1 and R@5 respectively. When contrasting these conclusions with our study, there was congruence at R@1 with style 2a, though discrepancies arose at R@5. A deeper inspection of Figure 10 contradicts Brate et al.'s [1] assertion, placing style 2b as the fourth best-performing original style at R@1. A notable observation from Table 15 is that the mean differences in our study for BERT substantially exceed those reported by Brate et al. [1]. Although Brate et al. [1] documented ranges of 0.006-0.161 and 0.062-0.515 for original prompts at R@1 and R@5 respectively, our analysis presents slightly wider ranges of 0.004-0.298 and 0.103-0.555. BERT's performance in this study surpassed its performance in the original paper.

The observed discrepancies between the findings of the present study and those of Brate et al. [1] can be attributed to the methodological choice of swapping closely predicted words corresponding to genres, as elaborated in Section 4.4 Large Language Model Probing. An examination of Table 16 provides insights into the distribution of proximately predicted words amongst the LLMs. Specifically, BERT recorded an average of 1,584 replacements per prompt style at R@10. However, as previously mentioned, these changes were necessary to allow BART and ALBERT to show significant results, as well as to even the playing field between the LLMs that generated similes of the genres, phrased slightly differently to how they are phrased in the original dataset, despite having the same meaning (for example "animated" and "animation", as shown in Table 10).

One plausible explanation for the divergent outcomes in the best-performing 'b' styles lies in the methodological choice to omit the initial "and" subsequent to the movie title in the 'b' prompts, as delineated in Section 4.3 Prompt Engineering. Another contributing factor is the discrepancy in the number of movies retained post-dataset cleaning. In our refined dataset, we retained 8,812 movies with comprehensive KG properties, in contrast to the 9,596 movies used by Brate et al. [1]. Such variances can be attributed to the nuances in dataset-cleaning procedures, potentially introducing a marginal bias in our dataset.

A noteworthy observation is the markedly inferior performance of style 6a relative to other original prompts across all R@n measures. This suggests that this particular style may activate an inherent anomaly within BERT, leading to a majority of its predictions being incorrect. Specifically, prompt 6a incorporates the editor as the terminal KG property in the prompt, delineated by commas. Similarly, style 9a, which designates the country of origin as the concluding KG property, also underperformed in comparison to most analogous styles. Such findings insinuate that appending specific attributes like the editor or country of origin, succeeded by the mask token excerpt and demarcated by commas, might profoundly alter BERT's interpretative framework of the sentence. Interestingly, this transformative effect was absent in the respective 'b' styles.

RoBERTa's superior performance in comparison to other LLMs across a majority of prompt styles implies that it exhibits a distinct advantage in the cloze-style genre task. However, specific prompt formulations seem to destabilize its consistency. Notably, RoBERTa surpasses other LLMs in the naturally-worded prompts 25-36, with the sole exception being ALBERT's response to prompt 26 across all R@n. Given that RoBERTa is a refined evolution of the BERT architecture, its outperformance of both BERT and ALBERT is anticipated. Yet, delineating the performance variance between RoBERTa and BART remains complex. It is crucial to underscore that BART operates as an auto-regressive model, a characteristic distinct from the other three LLMs.

As highlighted earlier, RoBERTa demonstrated notably suboptimal performance with styles 4a and 6a across all R@n measures. These styles incorporate the screenwriter and editor, respectively, with each delineated by commas. When examining the range of KG properties presented in Table 6, one might infer that the roles of screenwriter and editor are arguably the least recognized or prominent KG properties associated with films [70]. Relative to other listed KG properties, their potential infrequency in RoBERTa's training datasets could account for the observed dip in its efficacy.

Comparing original prompt performances, Brate et al. [1] documented RoBERTa's recall ranges as 0.004-0.210 (R@1) and 0.031-0.576 (R@5). This research, however, delineated broader ranges: 0.024-0.419 (R@1) and 0.131-0.746 (R@5). One should note the more pronounced differentiation in recall scores between our study and Brate et al. [1] when evaluating RoBERTa as opposed to BERT. Both investigations pinpointed 9b as the best-performing original prompt for R@5. Yet, at R@1, our analysis favoured 7b (recall of 0.419) over 9b (recall of 0.416) - a minuscule variance of 0.003. Hence, the two studies nearly converge in their assessments of ideal prompt styles for RoBERTa.

In contrast to BERT, the mean differences observed for RoBERTa between the current study and that by Brate et al. [1] are notably more consistent, particularly at R@5 as illustrated in Table 15. This can be attributed to the fact that, according to Table 16, RoBERTa averages 1085 replacements for a singular prompt style, whereas BERT records a higher average of 1781 replacements.

In Section 2.2.4 BART Large, it is highlighted that BART was pre-trained using an array of masked language modelling techniques, which includes the deletion of tokens or entire text spans. Given that functional words rank among the most prevalent in the English lexicon, the extensive use of such masked techniques can predispose an LLM to favor these words. Consequently, when predicting English words, there is an inherent proclivity of BART to gravitate towards functional words over more domain-specific terms, such as movie genres, due to its pre-training regime.

In earlier discussions, we observed that among BART's best-performing prompts were the naturally-worded prompts 25-36, with specific emphasis on prompts 31, 32, and 34 across all R@n metrics. Their T5-paraphrased and thesaurus-paraphrased counterparts (50, 61, and 64-66) mirrored this success. One plausible explanation lies in BART's training on datasets that likely encompass a greater degree of informal text, in contrast to BERT. For instance, BART's exposure to the OpenWebText corpus [50] - which was created from a range of webpages found in URLs from Reddit comments with more than 2 upvotes - would contain informal pieces of text that introduce BART to a wide array of casual, conversational, and sometimes colloquial language, which often includes various vernacular, slangs, abbreviations, and emojis. Furthermore, corpora like these adhere to a conversational context, where language is generally more interactive and dynamic, allowing LLMs trained on these corpora to understand more naturally-worded prompts.

The CC-News corpus [49], utilized in BART's training, encompasses millions of news articles. It is worth noting that, in contrast to Wikipedia's meticulous adherence to neutrality—"Articles must not take sides, but should explain the sides, fairly and without editorial bias. This applies to both what you say and how you say it"[45]—news articles, especially opinion columns, are much more likely to harbour more opinionated and polarizing content. In a similar vein, the STORIES corpus [51] captures the diverse and intricate nuances of storytelling, ranging from character dialogues, descriptions, emotions, to the unpredictability of plot developments. Such a broad spectrum of narrative techniques and styles equips BART with a richer understanding of human language, allowing it to generate more creative, context-aware, and engaging responses, mimicking the organic flow of storytelling.

In contrast, databases such as Wikipedia and BookCorpus, on which BERT is predominantly trained, inherently manifest a more formalized and systematized linguistic pattern. This results in a diminished variability in both tone and stylistic elements, making them more akin to the structured prompts 1-24 than the naturally-worded prompts 25-36. Moreover, such datasets often lack the dynamic, interactive context present in informal texts. This discrepancy offers a plausible explanation for the marked performance differential BART displays between prompts 1-24 and 25-36 in comparison to BERT. This also implies that BERT would perform better on the list-like original prompts compared to the naturally-worded ones, which is true in this study.

However, an anomaly in this hypothesis is presented by the performance of ALBERT, another LLM trained exclusively on Wikipedia and BookCorpus. Surprisingly, ALBERT demonstrates a superior efficacy with the naturally-worded prompts 25-36 across all R@n metrics compared to other prompts. Its subpar performance with the structured prompts 1-24 suggests that the challenges faced by ALBERT may stem more from inherent architectural limitations rather than solely from its training data.

[45] `https://en.wikipedia.org/wiki/Wikipedia:Neutral_point_of_view`

Considering that RoBERTa and BART shared identical training corpora, it provides a rationale for RoBERTa's enhanced performance on the naturally-worded prompts 25-36 in contrast to BERT and ALBERT, both exclusively trained on Wikipedia and BookCorpus. Furthermore, distinctions between BERT and RoBERTa, as well as between the subsequent LLMs, can be traced back to disparities in the sizes of their training datasets and the number of embedded transformer layers. Such differences precipitate the notable disparities observed in the mean differences across the best-performing prompts, as depicted in Figure 5.

Given ALBERT's design philosophy, which leans towards a more compact rendition of BERT, the observed decline in its performance across most of the original and custom prompt styles relative to its LLM counterparts is explicable. ALBERT's constrained diversity in training data impedes its capability to associate KG properties with specific movie genres. It should be noted, however, that ALBERT demonstrates superior performance with the naturally-worded styles 25-36 across all R@n metrics. In training a model on a more restricted corpus, greater emphasis might be placed on distinct linguistic features prevalent in naturally-worded prompts. This heightened focus could elucidate ALBERT's relative proficiency with certain prompts, indicating an alignment with its training paradigms. For instance, ALBERT outperforms all the other LLMs on prompt 26 across all R@n.

While the majority of the LLMs demonstrated commendable performance with styles 31 and 32, the results for their round-trip translations (styles 43-46) exhibited notably inferior efficacy. This observation underscores the possibility that the semantic integrity of these naturally-worded styles may not have been rigorously upheld during the translation process, culminating in diminished performance. Notably, styles 43 and 45, which correspond to the French translations of styles 31 and 32, ranked among the least proficient styles for most LLMs. Adding a layer of complexity, an assessment of styles 37-42 revealed a consistent trend where French translations generally outperformed their German counterparts, with BERT's performance serving as a salient exemplar. This suggests a non-uniform optimal language paradigm for round-trip translations, reinforcing the crucial role of nuanced prompt semantics in the efficacy of the translation process.

Table 18 lists the top 5 movies with the highest average recall scores for R@1. We can see that 4 of the 5 movies are comedy movies, the second-most labelled genre from the cleaned dataset, as seen in Table 5. Additionally, 3 of the highlighted films are encompassed within the horror genre, possessing overtly indicative titles such as "The Comedy of Terrors" and "The Last Horror Film". These unambiguous designations, enriched with semantically potent terms like "terrors" and manifestly "horror", enable the LLMs to render accurate genre predictions with relative ease. A case in point, "The Comedy of Terrors", wherein half of its title directly signifies its dual genres, stands out with an average recall score substantially elevated compared to all other movies in our dataset.

It is challenging to definitively identify the overall best prompt style between style 50 and style 9b, the two best-performing styles for the LLMs at various R@n, as seen in Table 12, showing significant mean differences between the unenriched prompts and these styles. This suggests that, at the very least, the ensemble methodology of naturally wording a prompt with the best-performing KG properties, followed by T5-paraphrasing said prompt, is one of the best prompt-generation techniques used in this paper.

The breakdown of the runtimes for each LLM is displayed in Table 17. BART exhibited the longest runtime of just under 11 hours, while BERT exhibited the shortest runtime of just under 4 hours. Surprisingly, despite ALBERT being a compressed version of BERT, its runtime was only 8 minutes shorter than RoBERTa. In fact, for some of the prompt groupings, such as the intermediate and custom ones, ALBERT's runtime was longer than RoBERTas.

In the next chapter, we will discuss the legal, social, ethical and professional issues faced when conducting this study.

	Original	**Intermediate**	**Custom**	**Translated**	**Paraphrased**	**Thesaurus**	**Total**
BERT	3 hrs 50 mins	0 hrs 42 mins	0 hrs 36 mins	1 hrs 24 mins	0 hrs 20 mins	0 hrs 11 mins	0 hrs 34 mins
RoBERTa Large	7 hrs 18 mins	1 hrs 40 mins	1 hrs 3 mins	2 hrs 31 mins	0 hrs 38 mins	0 hrs 19 mins	1 hrs 4 mins
BART Large	10 hrs 51 mins	1 hrs 38 mins	1 hrs 26 mins	3 hrs 52 mins	0 hrs 55 mins	0 hrs 26 mins	2 hrs 32 mins
ALBERT Large	7 hrs 10 mins	1 hrs 14 mins	1 hrs 12 mins	2 hrs 40 mins	0 hrs 38 mins	0 hrs 20 mins	1 hrs 3 mins

Table 17: Runtime comparison for different models and tasks.

Movie	**Genre(s)**	**Average Recall (3 SF)**
The Comedy of Terrors (1963)	Comedy Horror	0.596
The Last Horror Film (1982)	Comedy Horror	0.514
Crazy, Stupid, Love. (2011)	Comedy Drama Romance	0.467
Dr. Terror's House of Horrors (1965)	Horror	0.462
Monty Python's Life of Brian (1979)	Comedy	0.451

Table 18: Top 5 movies with the highest average recall scores across all LLMs and prompt styles at R@1.

6 Legal, Social, Ethical and Professional Issues

This chapter discusses the legal, social, ethical and professional issues faced during the implementation of this project.

The British Computer Societies' Code of Conduct[46] and Code of Good Practice[47] have been thoroughly considered and integrated throughout the entire project.

The Code of Conduct has been meticulously adhered to, ensuring that the project duly respects public health, security, privacy, and the well-being of both individuals and the environment. This careful adherence also ensures compliance with existing laws and upholds the prestige and good standing of the British Computer Society.

The Code of Good Practice has been judiciously followed, guaranteeing that the project aligns with existing regulations and standards pertinent to King's College London. The project contributes to contemporary advancements in the relevant speciality area, employing appropriate methodologies and tools. It contributes to public schooling whenever feasible, and it demonstrates an advanced understanding of current legislation, security advisories, and regulations. All reasonable measures have been taken to ensure that the project's output and any associated consequences pose no unacceptable risk to safety.

This project employs several LLMs trained on expansive corpora. In working with these LLMs, no identifiable data from corpora has been utilized or stored in the code or in the report. Moreover, this project processes a significant amount of potentially sensitive data - primarily the names of actors, directors, producers, composers, screenwriters, editors, and distribution companies. However, all the data processed within this project are already publicly available through platforms like Wikidata (and more commonly, Wikipedia), mitigating any potential issues.

This project has not been designed for commercial use. All textual references, figure and table origins, and website mentions have been clearly credited in the report. The main content of the project report and its related code are products of my own effort, with all external contributions duly acknowledged.

In the next chapter, we will discuss the conclusions and any possible future work of this study.

[46] https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/

[47] https://www.inf.ed.ac.uk/teaching/courses/pi/2013_2014/notes/1/cop.pdf

7 Conclusion

In the present study, we have extended the research pioneered by Brate et al. [1], who experimented with the potential of augmenting naive prompts with knowledge graph (KG) contextual details to bolster the performance of two large language models (LLMs) in a movie genre prediction task. By broadening our focus to encompass four distinct LLMs, our empirical findings reveal with statistical significance that the integration of contextual KG properties into prompts considerably elevates the performance of each LLM across various recall levels. This aligns with and corroborates the foundational conclusions drawn by Brate et al. [1].

Furthermore, our results, grounded in statistical analyses, denote that a subset of the prompts we devised - naturally-worded language, round-trip translations, and various paraphrasing methodologies - outperformed the original prompts constructed by Brate et al. [1] for all LLMs and recall levels, excluding RoBERTa large at recall@5 and recall@10. Our investigation also underscores the pivotal role of the meticulous construction of prompts supplemented with KG properties - even minute modifications in wording or punctuation markedly influence LLM performance. We surmise that the idealized prompt configuration for cloze-style assignments is contingent upon the underlying architecture of a particular LLM and the level of formality of its training corpora. For this specific cloze-style genre prediction task, we found that the best prompt generation technique was an ensemble of naturally wording a prompt based on its best-performing KG properties, followed by rephrasing said prompt using FLAN-T5 [57].

We believe the findings of our study hold profound implications for the utilization and optimization of LLMs in practical applications. As the era of big data and LLMs progresses, the ability to finetune LLMs with carefully structured prompts can significantly enhance the capabilities of recommendation systems, content classifiers, and many other LLM-driven solutions. This is especially pertinent in sectors like media and entertainment, where genre classification plays a pivotal role. We have not only demonstrated the tangible benefits of prompt engineering, but also elucidated the nuanced intricacies of prompt structure that can make or break the performance of an LLM. We urge the research community to explore the potential of integrating KGs into prompt design, ensuring a more contextually aware and precise LLM system. Moving forward, as we expand our understanding of prompt engineering, we foresee a future where LLMs are fine-tuned to specific tasks with unprecedented accuracy, paving the way for advancements in media recommendation, genre classification, and a myriad of related tasks.

7.1 Future Work

Future research can based on the findings of this research are manifold. Echoing the suggestions of Brate et al. [1], subsequent studies could be extended to encompass alternative forms of media such as literary works or musical compositions, providing a broader empirical landscape. Further studies could aim to substantiate the findings presented both in the current study and those of Brate et al. [1]. Future work could also explore the differences between the sizes of LLMs, such as between RoBERTa, RoBERTa Large, RoBERTa Extra Large, etc., using the methodologies proposed in this paper to ascertain the differences in performances between these LLMs with the cloze-style movie genre prediction task.

An additional research trajectory could involve crafting prompt styles that systematically encompass every feasible amalgamation of KG properties. This would offer insights into the optimal combination of KG attributes, marking a departure from our employed methodology wherein KG properties were incrementally integrated into prompts as opposed to an exhaustive approach.

Subsequent research endeavours might consider leveraging the methodologies employed in this investigation in an ensemble framework to ascertain if it augments LLM efficacy. For instance, prompts could undergo a process where they are initially crafted in natural language, subsequently translated in a round-trip manner, paraphrased, and then presented to the LLMs. It might also be worthwhile to diversify the selection of LLMs for prompt generation, as opposed to the sole reliance on MarianMT for translation and FLAN-T5 [57] for paraphrasing. The thesaurus-based paraphrasing technique from this investigation could be enhanced to methodically paraphrase prompts by integrating an exhaustive list of synonyms, moving beyond the current scope of generating only three alternative thesaurus-paraphrased prompts.

Future research trajectories could also depart from the zero-shot learning paradigm employed in this study. They might explore other methodologies, including the automatic prompt engineering techniques delineated in Section 2.3 Related Work. Such investigations can rigorously assess an array of prompt modalities and determine whether the most efficacious prompts are those crafted manually by experts or generated autonomously by other LLMs. Further experimental designs can involve training LLMs on labelled movie genre datasets, aiming to elucidate the specific influence of KG properties during training on LLM performance within a cloze-style task.

References

[1] R. Brate, M.-H. Dang, F. Hoppe, Y. He, A. Meroño-Peñuela, and V. Sadashivaiah, "Improving language model predictions via prompts enriched with knowledge graphs," publisher: CEUR-WS.org. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000151291

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need." [Online]. Available: http://arxiv.org/abs/1706.03762

[3] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." [Online]. Available: http://arxiv.org/abs/1910.13461

[4] H. J. Levesque, "The winograd schema challenge."

[5] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," vol. 5, no. 4, pp. 1–19. [Online]. Available: https://dl.acm.org/doi/10.1145/2827872

[6] OpenAI, "GPT-4 technical report," issue: arXiv:2303.08774. [Online]. Available: http://arxiv.org/abs/2303.08774

[7] D. M. Katz, M. J. Bommarito, S. Gao, and P. Arredondo, "GPT-4 passes the bar exam." [Online]. Available: https://www.ssrn.com/abstract=4389233

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[9] An important next step on our AI journey. [Online]. Available: https://blog.google/technology/ai/bard-google-ai-search-updates/

[10] Reinventing search with a new AI-powered microsoft bing and edge, your copilot for the web. [Online]. Available: https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/

[11] K. Hu, "ChatGPT sets record for fastest-growing user base - analyst note." [Online]. Available: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/

[12] D. Milmo, "ChatGPT reaches 100 million users two months after launch." [Online]. Available: https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app

[13] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" [Online]. Available: http://arxiv.org/abs/1909.01066

[14] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 5418–5426. [Online]. Available: https://aclanthology.org/2020.emnlp-main.437

[15] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with ChatGPT." [Online]. Available: http://arxiv.org/abs/2302.11382

[16] F. Petroni, P. Lewis, A. Piktus, T. Rocktäschel, Y. Wu, A. H. Miller, and S. Riedel, "How context affects language models' factual predictions." [Online]. Available: http://arxiv.org/abs/2005.04611

[17] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity." [Online]. Available: http://arxiv.org/abs/2104.08786

[18] T. Pham, T. Bui, L. Mai, and A. Nguyen, "Out of order: How important is the sequential order of words in a sentence in natural language understanding tasks?" in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, pp. 1145–1160. [Online]. Available: https://aclanthology.org/2021.findings-acl.98

[19] W. Yuan, G. Neubig, and P. Liu, "BARTScore: Evaluating generated text as text generation." [Online]. Available: http://arxiv.org/abs/2106.11520

[20] A. Mishra, U. Soni, A. Arunkumar, J. Huang, B. C. Kwon, and C. Bryan, "PromptAid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models." [Online]. Available: http://arxiv.org/abs/2304.01964

[21] C. Liu, T. Cohn, and L. Frermann, "Seeking clozure: Robust hypernym extraction from BERT with anchored prompts," in *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*. Association for Computational Linguistics, pp. 193–206. [Online]. Available: https://aclanthology.org/2023.starsem-1.18

[22] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?" [Online]. Available: http://arxiv.org/abs/1911.12543

[23] J. Mallinson, R. Sennrich, and M. Lapata, "Paraphrasing revisited with neural machine translation," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, pp. 881–893. [Online]. Available: https://aclanthology.org/E17-1083

[24] R. Sennrich, B. Haddow, and A. Birch, "Improving neural machine translation models with monolingual data," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 86–96. [Online]. Available: https://aclanthology.org/P16-1009

[25] A. Haviv, J. Berant, and A. Globerson, "BERTese: Learning to speak to BERT," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, pp. 3618–3623. [Online]. Available: https://aclanthology.org/2021.eacl-main.316

[26] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers." [Online]. Available: http://arxiv.org/abs/2211.01910

[27] W. L. Taylor, ""cloze procedure": A new tool for measuring readability," vol. 30, no. 4, pp. 415–433. [Online]. Available: http://journals.sagepub.com/doi/10.1177/107769905303000401

[28] A review: Knowledge reasoning over knowledge graph | elsevier enhanced reader. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/S0957417419306669?token=011D8DC2B81D48FAE36900B86D632D2B8B0E3080F2F81289A1E1CD53766EB974F035A114C716244935AE0F originRegion=eu-west-1&originCreation=20230420225715

[29] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, "Introduction: What is a knowledge graph?" in *Knowledge Graphs: Methodology, Tools and Selected Use Cases*, D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, Eds. Springer International Publishing, pp. 1–10. [Online]. Available: https://doi.org/10.1007/978-3-030-37439-6_1

[30] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs."

[31] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," vol. 34, no. 3, pp. 1–45. [Online]. Available: https://dl.acm.org/doi/10.1145/1567274.1567278

[32] RDF - semantic web standards. [Online]. Available: https://www.w3.org/RDF/

[33] F. Suchanek. YAGO: A large ontology from wikipedia and WordNet. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/S1570826808000437?token= DA55ED5874DD5B63F75F41233C18E5EB63FAABF8045616B106582016F542A6ED1E550E2CF19813F864DD2 originRegion=eu-west-1&originCreation=20230420230143

[34] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo, and G. Weikum, "YAGO2: exploring and querying world knowledge in time, space, context, and many languages," in *Proceedings of the 20th international conference companion on World wide web*. ACM, pp. 229–232. [Online]. Available: https://dl.acm.org/doi/10.1145/1963192.1963296

[35] T. Pellissier Tanon, G. Weikum, and F. Suchanek, "YAGO 4: A reason-able knowledge base," in *The Semantic Web*, A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, Eds. Springer International Publishing, vol. 12123, pp. 583–596, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-030-49461-2_34

[36] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," vol. 57, no. 10, pp. 78–85. [Online]. Available: https://dl.acm.org/doi/10.1145/2629489

[37] D. Vrandečić, "Wikidata: a new platform for collaborative data collection," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12 Companion. Association for Computing Machinery, pp. 1063–1064. [Online]. Available: https://doi.org/10.1145/2187980.2188242

[38] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon, "Domain-specific language model pretraining for biomedical natural language processing," vol. 3, no. 1, pp. 1–23. [Online]. Available: http://arxiv.org/abs/2007.15779

[39] Y. Shen. ChatGPT and other large language models are double-edged swords. [Online]. Available: https://pubs.rsna.org/doi/epdf/10.1148/radiol.230163

[40] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," p. 3605943. [Online]. Available: https://dl.acm.org/doi/10.1145/3605943

[41] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding." [Online]. Available: http://arxiv.org/abs/1810.04805

[42] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach." [Online]. Available: http://arxiv.org/abs/1907.11692

[43] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," vol. 10, no. 2, pp. 1–37. [Online]. Available: https://dl.acm.org/doi/10.1145/3293318

[44] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. M. J. Wu, "A review of generalized zero-shot learning methods," vol. 45, no. 4, pp. 4051–4070, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[45] Z. Fu, T. A. Xiang, E. Kodirov, and S. Gong, "Zero-shot object recognition by semantic manifold distance," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 2635–2644. [Online]. Available: http://ieeexplore.ieee.org/document/7298879/

[46] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning." [Online]. Available: http://arxiv.org/abs/1904.05046

[47] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, pp. 353–355. [Online]. Available: http://aclweb.org/anthology/W18-5446

[48] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books." [Online]. Available: http://arxiv.org/abs/1506.06724

[49] J. Mackenzie, R. Benham, M. Petri, J. R. Trippas, J. S. Culpepper, and A. Moffat, "CC-news-en: A large english news corpus," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. ACM, pp. 3077–3084. [Online]. Available: https://dl.acm.org/doi/10.1145/3340531.3412762

[50] A. Gokaslan and V. Cohen, "OpenWebText corpus." [Online]. Available: http://Skylion007.github.io/OpenWebTextCorpus

[51] T. H. Trinh and Q. V. Le, "A simple method for commonsense reasoning." [Online]. Available: http://arxiv.org/abs/1806.02847

[52] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations." [Online]. Available: http://arxiv.org/abs/1909.11942

[53] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention." [Online]. Available: http://arxiv.org/abs/2006.03654

[54] L. Yang, S. Zhang, L. Qin, Y. Li, Y. Wang, H. Liu, J. Wang, X. Xie, and Y. Zhang, *GLUE-X: Evaluating Natural Language Understanding Models from an Out-of-distribution Generalization Perspective.*

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

[56] M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. F. Aji, N. Bogoychev, A. F. T. Martins, and A. Birch, "Marian: Fast neural machine translation in c++." [Online]. Available: http://arxiv.org/abs/1804.00344

[57] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models." [Online]. Available: http://arxiv.org/abs/2210.11416

[58] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer." [Online]. Available: http://arxiv.org/abs/1910.10683

[59] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators." [Online]. Available: http://arxiv.org/abs/2003.10555

[60] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding." [Online]. Available: http://arxiv.org/abs/1906.08237

[61] G. Penha and C. Hauff, "What does BERT know about books, movies and music? probing BERT for conversational recommendation," in *Fourteenth ACM Conference on Recommender Systems.* ACM, pp. 388–397. [Online]. Available: https://dl.acm.org/doi/10.1145/3383313.3412249

[62] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing." [Online]. Available: http://arxiv.org/abs/2107.13586

[63] P. Denny, V. Kumar, and N. Giacaman, "Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language." [Online]. Available: http://arxiv.org/abs/2210.15157

[64] M. Wermelinger, "Using GitHub copilot to solve simple programming problems," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* ACM, pp. 172–178. [Online]. Available: https://dl.acm.org/doi/10.1145/3545945.3569830

[65] L. Ruis, A. Khan, S. Biderman, S. Hooker, T. Rocktäschel, and E. Grefenstette, "Large language models are not zero-shot communicators." [Online]. Available: http://arxiv.org/abs/2210.14986

[66] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "ERNIE: Enhanced language representation with informative entities." [Online]. Available: http://arxiv.org/abs/1905.07129

[67] L. He, S. Zheng, T. Yang, and F. Zhang, "KLMo: Knowledge graph enhanced pretrained language model with fine-grained relationships," in *Findings of the Association for Computational Linguistics: EMNLP 2021.* Association for Computational Linguistics, pp. 4536–4542. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.384

[68] Stack overflow developer survey 2023. [Online]. Available: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023

[69] S. Cass. Top programming languages 2022 - IEEE spectrum. [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2022

[70] W. Goldman, *Adventures in the Screen Trade: A Personal View of Hollywood and Screenwriting*, repr ed. Abacus.

A Appendix: Tables and Figures

```
SELECT
  ?castLabel ?directorLabel ?producerLabel ?screenwriterLabel ?composerLabel
  ?editorLabel ?year ?distributorLabel ?countryLabel
WHERE {{
  ?film wdt:P31 wd:Q11424 .
  ?film rdfs:label ?label .
  FILTER(LANG(?label) = "en") .
  FILTER(CONTAINS(?label, "{movie_title}")) .
  OPTIONAL {{ ?film wdt:P161 ?cast . }}
  OPTIONAL {{ ?film wdt:P57 ?director . }}
  OPTIONAL {{ ?film wdt:P162 ?producer . }}
  OPTIONAL {{ ?film wdt:P58 ?screenwriter . }}
  OPTIONAL {{ ?film wdt:P86 ?composer . }}
  OPTIONAL {{ ?film wdt:P1040 ?editor . }}
  OPTIONAL {{ ?film wdt:P577 ?year . }}
  OPTIONAL {{ ?film wdt:P750 ?distributor . }}
  OPTIONAL {{ ?film wdt:P495 ?country . }}
  SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }}
}}
```

Figure 6: SPARQL query used to retrieve Wikidata properties used in this study.

Prompt	Description
1b	Life of Pi is a movie starring Adil Hussain and of the genre [MASK].
2b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and of the genre [MASK].
3b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and of the genre [MASK].
4b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and of the genre [MASK].
5b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and music by Mychael Danna and of the genre [MASK].
6b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and music by Mychael Danna and edited by Tim Squyres and of the genre [MASK].
7b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and music by Mychael Danna and edited by Tim Squyres and released in 2012 and of the genre [MASK].
8b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and music by Mychael Danna and edited by Tim Squyres and released in 2012 and distributed by InterCom and of the genre [MASK].
9b	Life of Pi is a movie starring Adil Hussain and directed by Ang Lee and produced by Ang Lee and screenwriter David Magee and music by Mychael Danna and edited by Tim Squyres and released in 2012 and distributed by InterCom and originating from United States of America and of the genre [MASK].
10b	The movie Life of Pi starring Adil Hussain and of the genre [MASK].
11b	The movie Life of Pi directed by Ang Lee and of the genre [MASK].
12b	The movie Life of Pi released in 2012 and of the genre [MASK].
13b	The movie Life of Pi originating from United States of America and of the genre [MASK].
14b	The movie Life of Pi starring Adil Hussain and directed by Ang Lee and of the genre [MASK].
15b	The movie Life of Pi starring Adil Hussain and released in 2012 and of the genre [MASK].
16b	The movie Life of Pi starring Adil Hussain and originating from United States of America and of the genre [MASK].
17b	The movie Life of Pi directed by Ang Lee and released in 2012 and of the genre [MASK].
18b	The movie Life of Pi directed by Ang Lee and originating from United States of America and of the genre [MASK].
19b	The movie Life of Pi released in 2012 and originating from United States of America and of the genre [MASK].
20b	The movie Life of Pi starring Adil Hussain and directed by Ang Lee and released in 2012 and of the genre [MASK].
21b	The movie Life of Pi starring Adil Hussain and directed by Ang Lee and originating from United States of America and of the genre [MASK].
22b	The movie Life of Pi starring Adil Hussain and released in 2012 and originating from United States of America and of the genre [MASK].
23b	The movie Life of Pi directed by Ang Lee and released in 2012 and originating from United States of America and of the genre [MASK].
24b	The movie Life of Pi starring Adil Hussain and directed by Ang Lee and released in 2012 and originating from United States of America and of the genre [MASK].

Table 19: A list of all of the prompt styles 1b-24b used in this paper, utilizing the movie ”Life of Pi” (2012) for illustrative purposes. Successive KG properties introduced, in contrast to the preceding row, are emphasized in red.

KG Property	Description
Cast	Life of Pi is a movie starring Adil Hussain and of the genre [MASK].
Director	Life of Pi is a movie directed by Ang Lee and of the genre [MASK].
Producer	Life of Pi is a movie produced by Ang Lee and of the genre [MASK].
Screenwriter	Life of Pi is a movie screenwriter David Magee and of the genre [MASK].
Composer	Life of Pi is a movie music by Mychael Danna and of the genre [MASK].
Editor	Life of Pi is a movie edited by Tim Squyres and of the genre [MASK].
Year	Life of Pi is a movie released 2012 and of the genre [MASK].
Distributor	Life of Pi is a movie distributed by InterCom and of the genre [MASK].
Country	Life of Pi is a movie originating from United States of America and of the genre [MASK].

Table 20: A list of all of the intermediate 'b' prompt styles used in this paper, utilizing the movie "Life of Pi" (2012) for illustrative purposes. KG properties and their labels are emphasized in red.

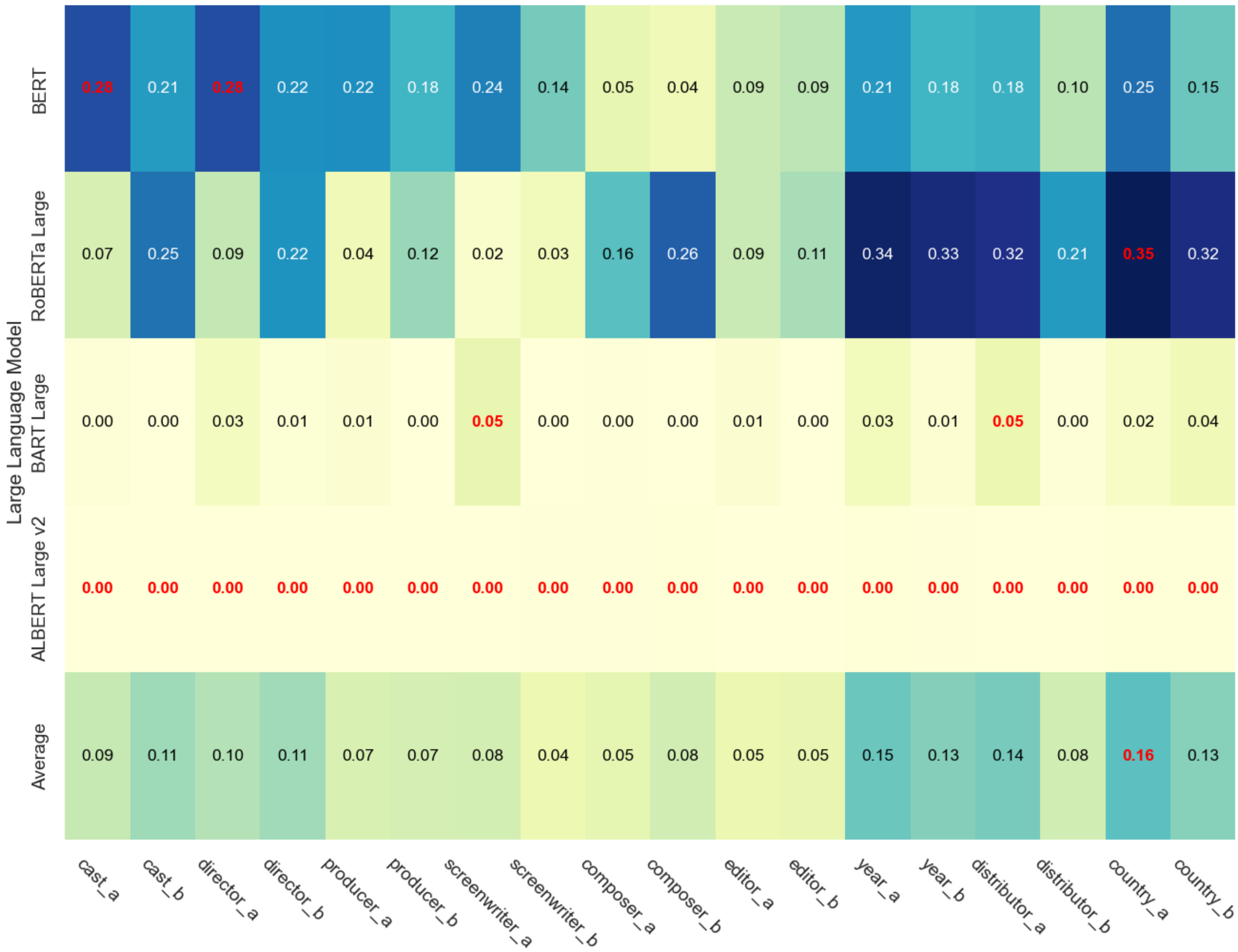

Figure 7: Average R@1 accuracies for each LLM and intermediate prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

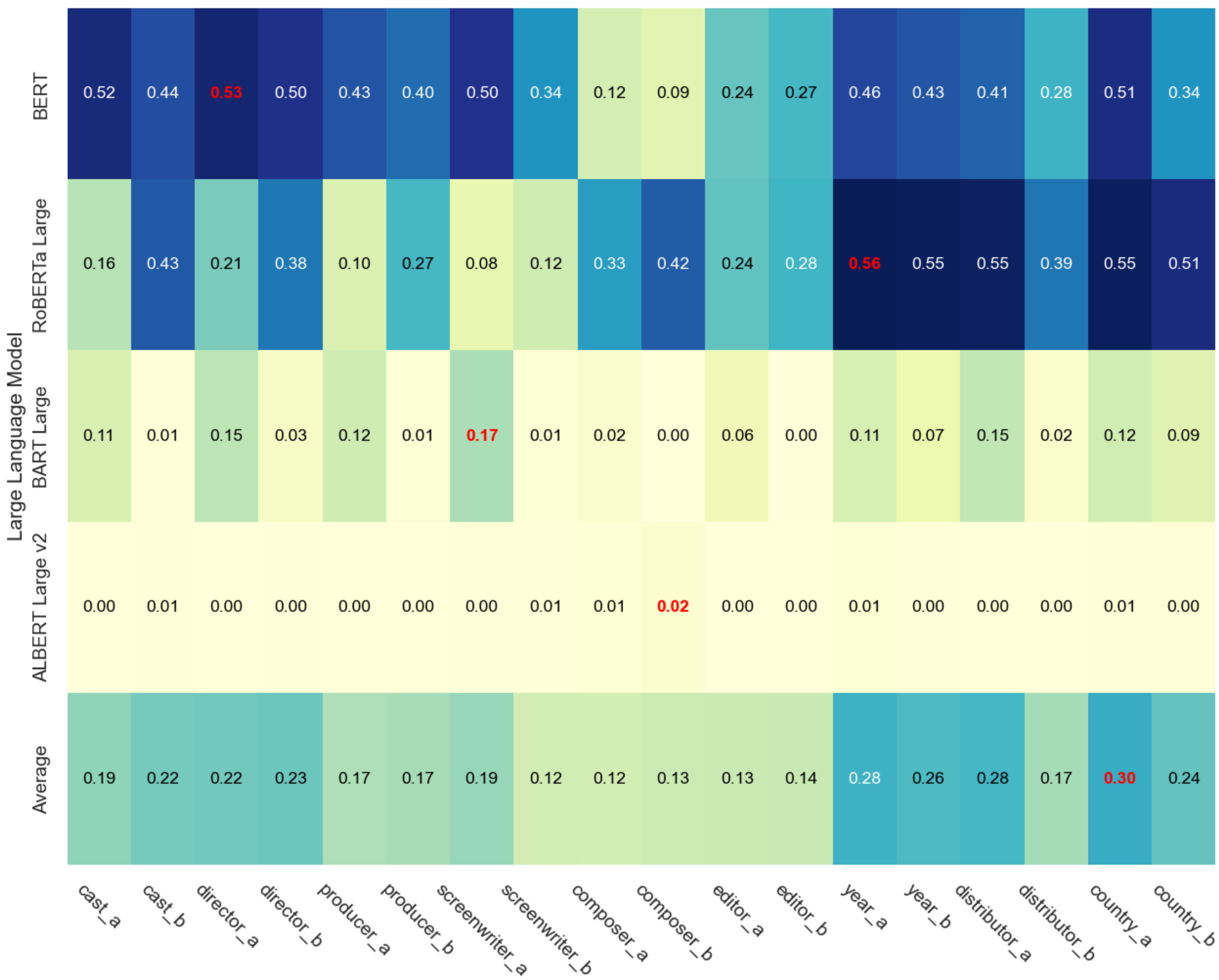

Figure 8: Average R@5 accuracies for each LLM and intermediate prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

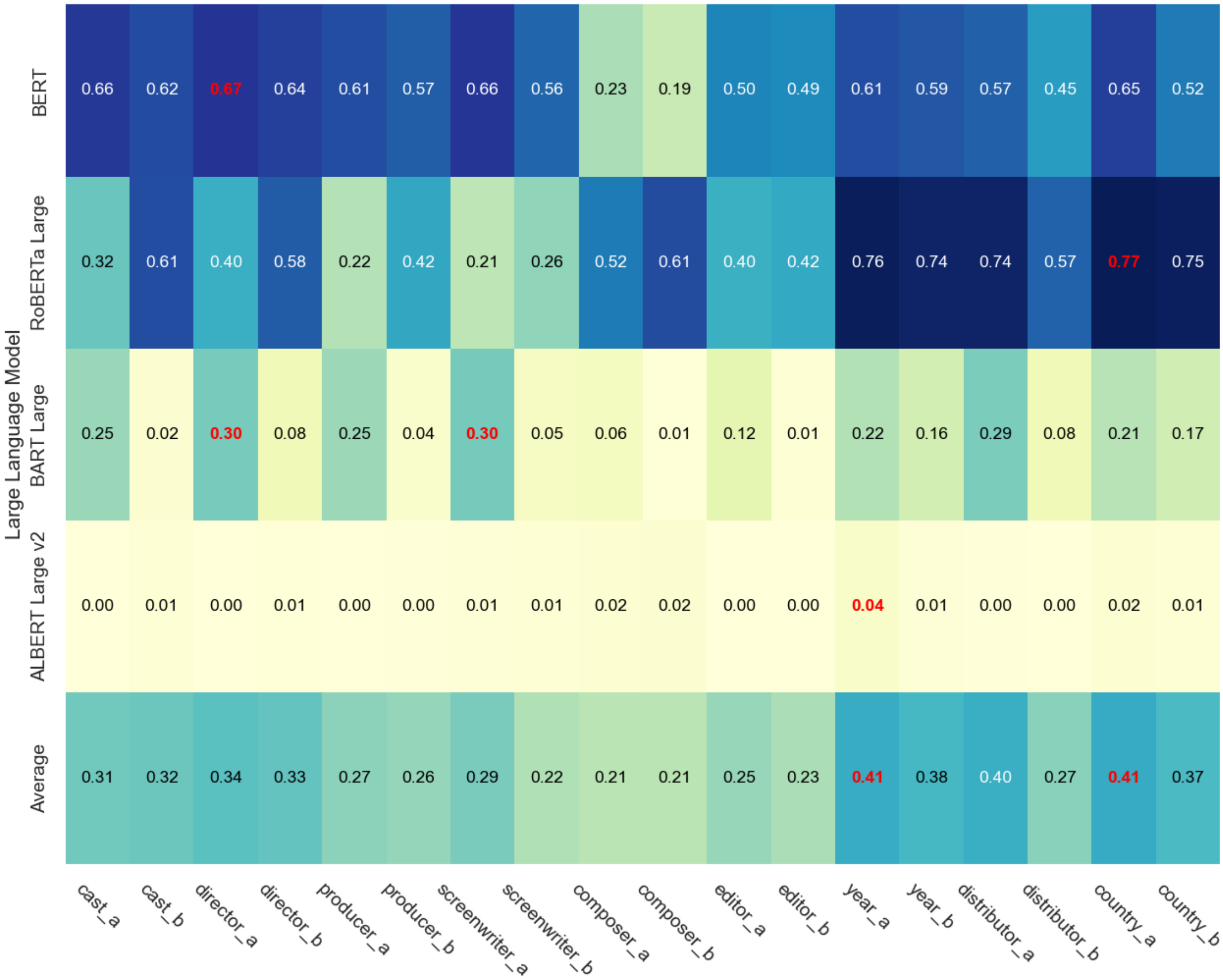

Figure 9: Average R@10 accuracies for each LLM and intermediate prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

Prompt	Description
52	Life of Pi is a film, featuring Adil Hussain, controlled by Ang Lee, of the genre [MASK].
53	Life of Pi is a flick, performing Adil Hussain, supervised by Ang Lee, of the genre [MASK].
54	Life of Pi is a picture, appearing Adil Hussain, guided by Ang Lee, of the genre [MASK].
55	Life of Pi is a film featuring Adil Hussain and controlled by Ang Lee and made by Ang Lee and scriptwriter David Magee and melody by Mychael Danna and corrected by Tim Squyres and announced 2012 and is of the genre [MASK].
56	Life of Pi is a flick performing Adil Hussain and supervised by Ang Lee and created by Ang Lee and playwright David Magee and harmony by Mychael Danna and modified by Tim Squyres and distributed 2012 and is of the genre [MASK].
57	Life of Pi is a picture appearing Adil Hussain and guided by Ang Lee and formed by Ang Lee and scripter David Magee and tune by Mychael Danna and revised by Tim Squyres and issued 2012 and is of the genre [MASK].
58	Life of Pi is a film featuring Adil Hussain and controlled by Ang Lee and made by Ang Lee and scriptwriter David Magee and melody by Mychael Danna and corrected by Tim Squyres and announced 2012 and allocated by InterCom and arising from United States of America and is of the genre [MASK].
59	Life of Pi is a flick performing Adil Hussain and supervised by Ang Lee and created by Ang Lee and playwright David Magee and harmony by Mychael Danna and modified by Tim Squyres and distributed 2012 and alloted by InterCom and developing from United States of America and is of the genre [MASK].
60	Life of Pi is a picture appearing Adil Hussain and guided by Ang Lee and formed by Ang Lee and scripter David Magee and tune by Mychael Danna and revised by Tim Squyres and issued 2012 and dispensed by InterCom and growing from United States of America and is of the genre [MASK].
61	A movie announced in 2012 out of United States of America, Life of Pi shows Adil Hussain and lapses into the [MASK] genre below the management of Ang Lee.
62	A flick distributed in 2012 arising out of United States of America, Life of Pi displays Adil Hussain and drifts into the [MASK] genre beneath the administration of Ang Lee.
63	A picture issued in 2012 coming out of United States of America, Life of Pi exhibits Adil Hussain and resorts the [MASK] genre underneath the government of Ang Lee.
64	Life of Pi, a masterwork in the [MASK] genre out of 2012, shows Ang Lee's invention and United States of America's lifestyle, featuring Adil Hussain.
65	Life of Pi, a coup in the [MASK] genre arising out of 2012, depicts Ang Lee's creativity and United States of America's customs, performing Adil Hussain.
66	Life of Pi, a classic in the [MASK] genre coming out of 2012, characterizes Ang Lee's inventiveness and United States of America's traditions, appearing Adil Hussain.

Table 21: A list of all of the thesaurus-paraphrased prompt styles 52-66 used in this paper, utilizing the movie "Life of Pi" (2012) for illustrative purposes.

LLM	Mask Token Style
BERT	[MASK]
RoBERTa large	<mask>
BART large	<mask>
ALBERT large v2	[MASK]

Table 22: LLMs and their mask styles.

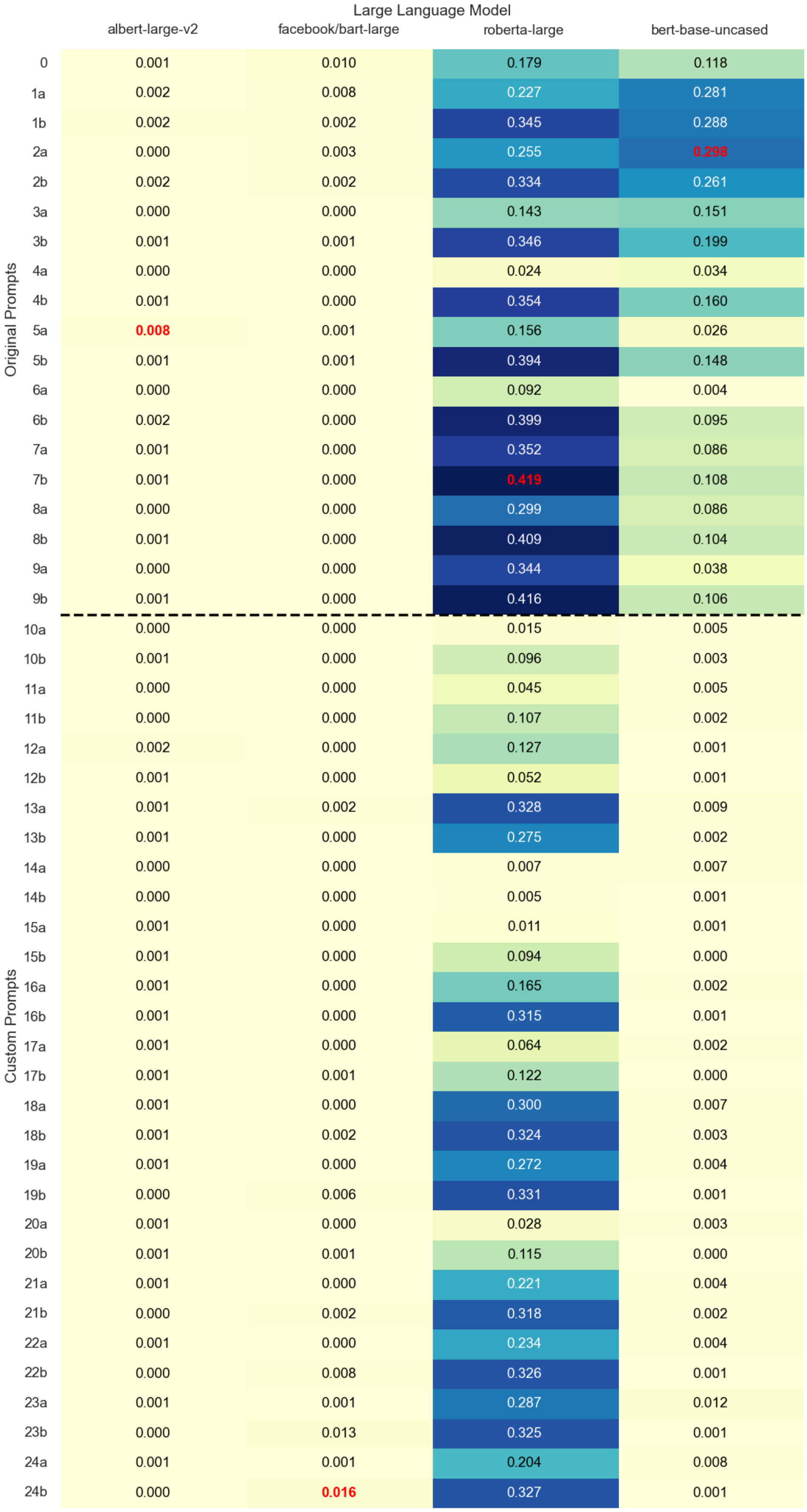

Figure 10: Average R@1 accuracy for each LLM and original and custom prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

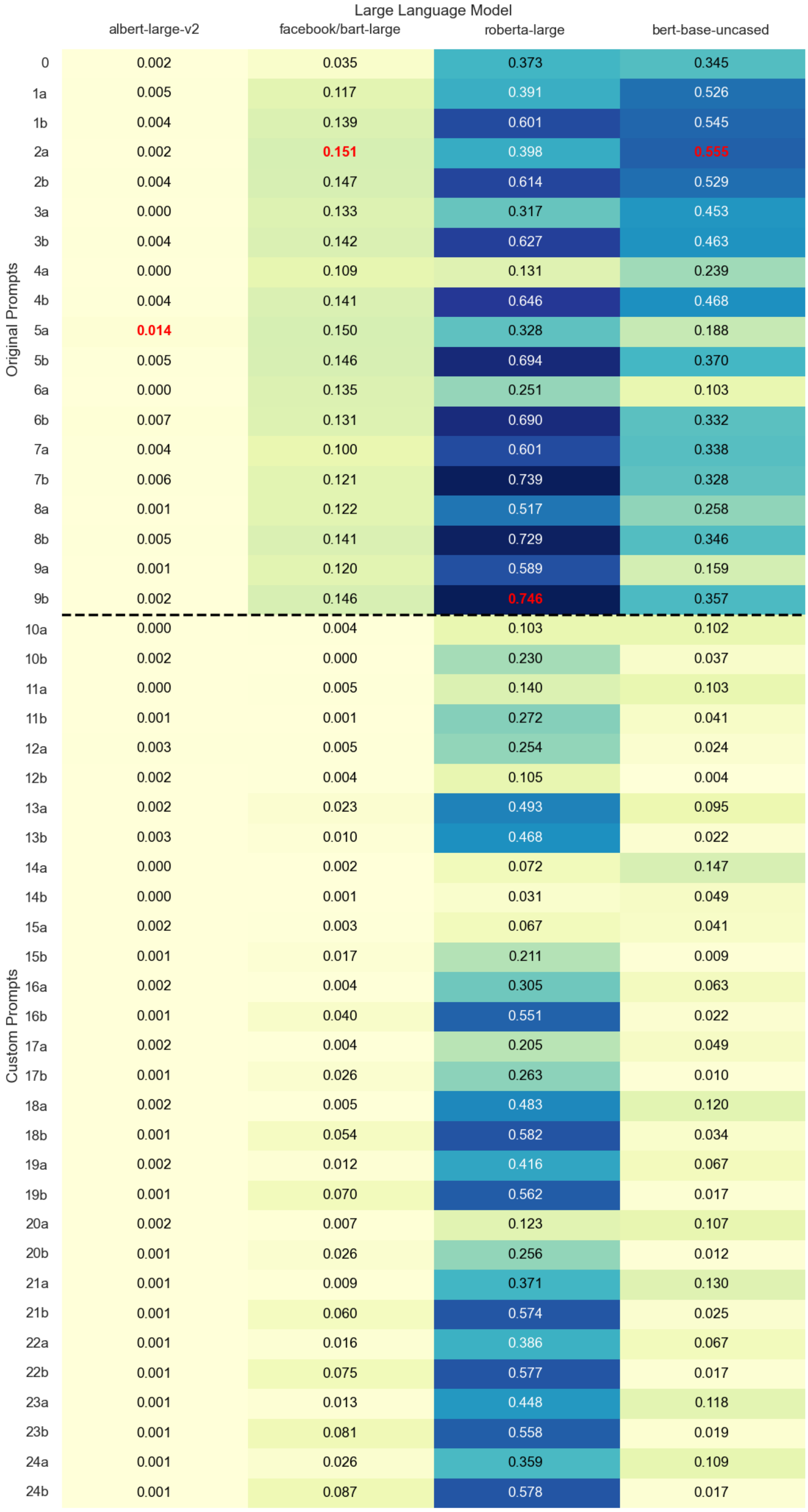

Figure 11: Average R@5 accuracies for each LLM and original and custom prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

		Large Language Model			
		albert-large-v2	facebook/bart-large	roberta-large	bert-base-uncased
Original Prompts	0	0.006	0.077	0.559	0.503
	1a	0.017	0.235	0.599	0.668
	1b	0.013	0.234	0.792	0.675
	2a	0.006	**0.280**	0.629	**0.689**
	2b	0.017	0.252	0.777	0.666
	3a	0.002	0.241	0.507	0.643
	3b	0.017	0.243	0.790	0.638
	4a	0.000	0.198	0.270	0.525
	4b	0.016	0.243	0.794	0.622
	5a	0.018	0.279	0.499	0.452
	5b	0.024	0.244	0.831	0.569
	6a	0.001	0.251	0.388	0.348
	6b	**0.034**	0.214	0.827	0.575
	7a	0.011	0.163	0.761	0.566
	7b	0.022	0.199	0.856	0.576
	8a	0.002	0.207	0.706	0.471
	8b	0.016	0.238	0.848	0.549
	9a	0.005	0.199	0.768	0.321
	9b	0.010	0.245	**0.862**	0.571
Custom Prompts	10a	0.001	0.012	0.210	0.319
	10b	0.004	0.000	0.370	0.138
	11a	0.000	0.014	0.281	0.335
	11b	0.002	0.003	0.433	0.197
	12a	0.007	0.018	0.432	0.142
	12b	0.003	0.014	0.195	0.048
	13a	0.004	0.055	0.720	0.291
	13b	0.004	0.031	0.696	0.070
	14a	0.000	0.005	0.164	0.387
	14b	0.000	0.003	0.104	0.259
	15a	0.004	0.010	0.159	0.163
	15b	0.002	0.042	0.352	0.135
	16a	0.004	0.015	0.490	0.209
	16b	0.003	0.075	0.757	0.070
	17a	0.005	0.009	0.366	0.257
	17b	0.001	0.059	0.432	0.170
	18a	0.004	0.014	0.701	0.325
	18b	0.002	0.101	0.772	0.122
	19a	0.004	0.032	0.650	0.204
	19b	0.002	0.125	0.757	0.053
	20a	0.003	0.016	0.258	0.305
	20b	0.002	0.060	0.416	0.202
	21a	0.003	0.028	0.585	0.326
	21b	0.002	0.117	0.756	0.070
	22a	0.003	0.038	0.612	0.204
	22b	0.002	0.147	0.754	0.048
	23a	0.003	0.035	0.693	0.283
	23b	0.002	0.165	0.743	0.054
	24a	0.002	0.051	0.600	0.255
	24b	0.001	0.169	0.750	0.041

Figure 12: Average R@10 accuracies for each LLM and original and custom prompt style. Values highlighted in red are the highest-performing prompt styles for each LLM.

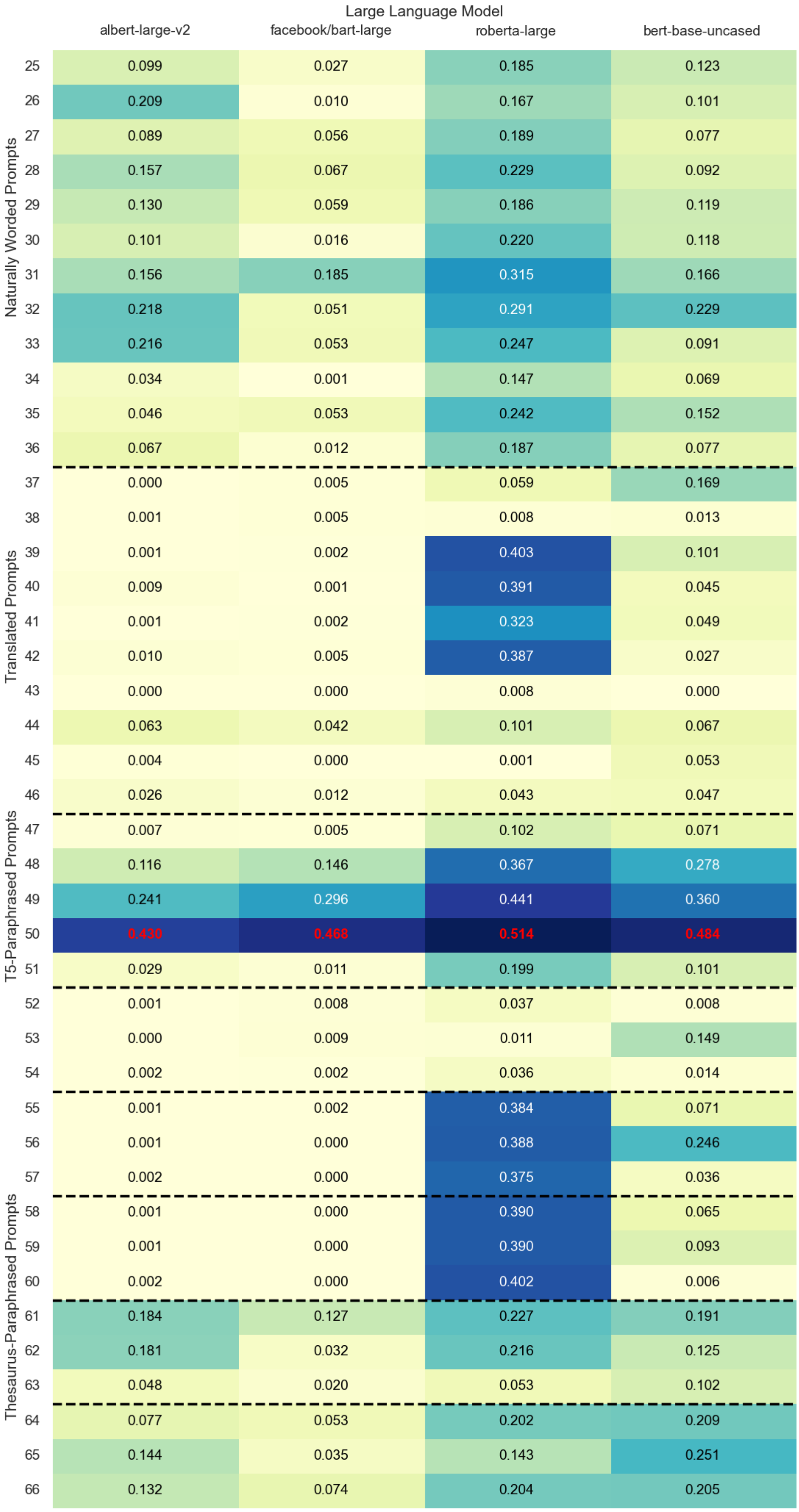

Figure 13: Average R@1 accuracy for each LLM and prompt styles 25-66. Values highlighted in red are the highest-performing prompt styles for each LLM.

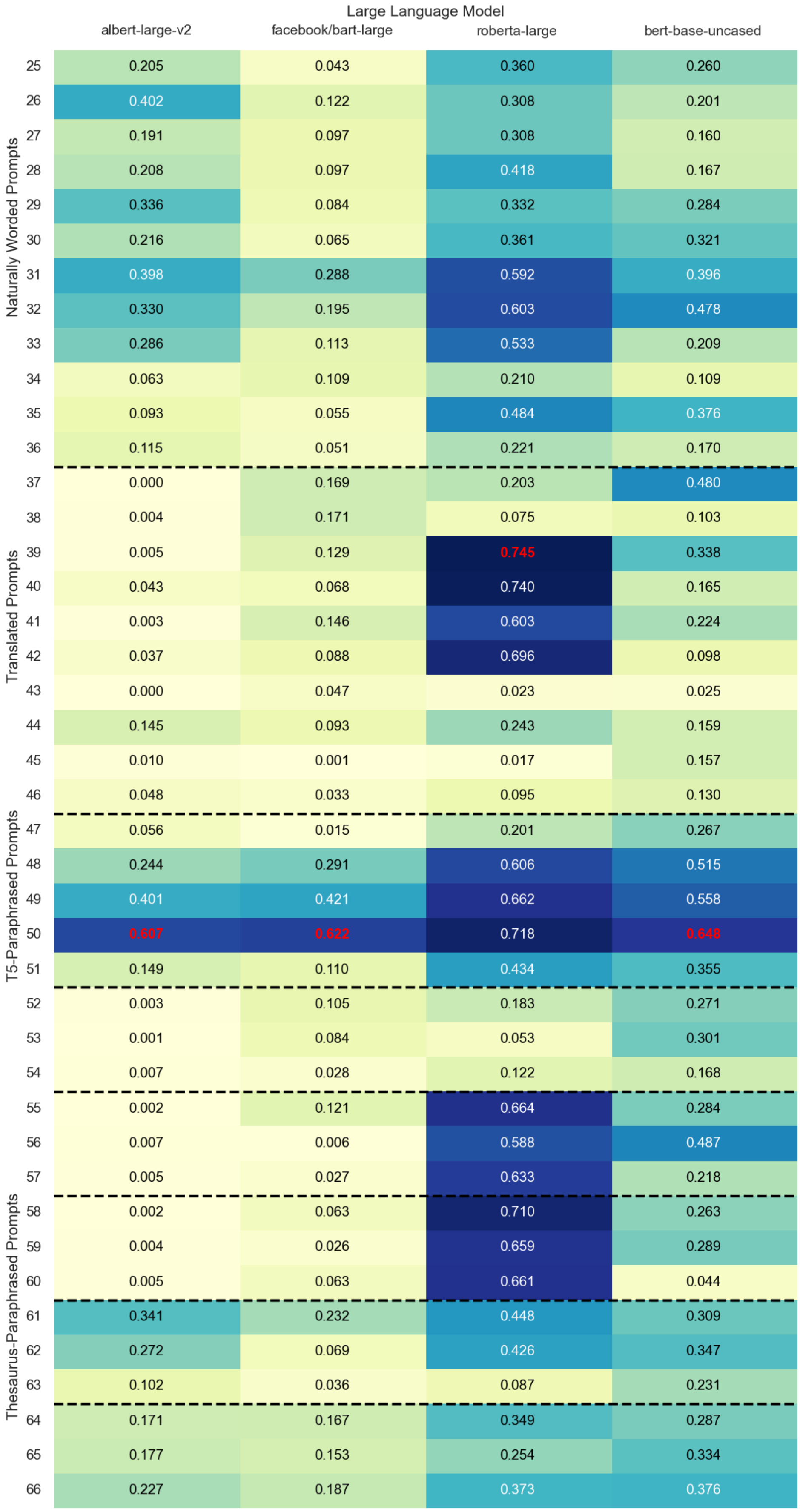

Figure 14: Average R@5 accuracies for each LLM and prompt styles 25-66. Values highlighted in red are the highest-performing prompt styles for each LLM.

		Large Language Model			
		albert-large-v2	facebook/bart-large	roberta-large	bert-base-uncased
Naturally Worded Prompts	25	0.311	0.061	0.564	0.424
	26	0.585	0.197	0.481	0.373
	27	0.314	0.146	0.487	0.247
	28	0.297	0.176	0.640	0.290
	29	0.517	0.133	0.539	0.446
	30	0.376	0.155	0.563	0.501
	31	0.636	0.452	0.829	0.531
	32	0.558	0.315	0.794	0.735
	33	0.519	0.185	0.716	0.406
	34	0.125	0.291	0.313	0.177
	35	0.243	0.100	0.747	0.575
	36	0.250	0.085	0.332	0.298
Translated Prompts	37	0.002	0.358	0.380	0.641
	38	0.016	0.435	0.202	0.295
	39	0.029	0.229	0.841	0.587
	40	0.118	0.143	0.826	0.302
	41	0.016	0.282	0.685	0.420
	42	0.101	0.168	0.818	0.175
	43	0.000	0.164	0.059	0.093
	44	0.232	0.176	0.380	0.237
	45	0.023	0.005	0.085	0.258
	46	0.078	0.060	0.170	0.226
T5-Paraphrased Prompts	47	0.166	0.036	0.369	0.494
	48	0.374	0.437	0.747	0.675
	49	0.537	0.558	0.792	0.723
	50	0.741	0.761	0.854	0.799
	51	0.291	0.253	0.588	0.541
	52	0.010	0.211	0.329	0.534
	53	0.004	0.197	0.127	0.464
	54	0.015	0.073	0.249	0.310
Thesaurus-Paraphrased Prompts	55	0.008	0.226	0.812	0.578
	56	0.032	0.041	0.773	0.625
	57	0.012	0.055	0.787	0.368
	58	0.008	0.147	0.843	0.536
	59	0.018	0.136	0.792	0.486
	60	0.012	0.145	0.806	0.127
	61	0.512	0.299	0.648	0.442
	62	0.331	0.132	0.604	0.453
	63	0.189	0.065	0.148	0.375
	64	0.250	0.258	0.541	0.412
	65	0.245	0.243	0.427	0.526
	66	0.322	0.288	0.599	0.620

Figure 15: Average R@10 accuracies for each LLM and prompt styles 25-66. Values highlighted in red are the highest-performing prompt styles for each LLM.

Genre	BERT	RoBERTa Large	BART Large	ALBERT Large v2
Action	0.014	0.033	0.061	0.015
Adventure	0.02	0.012	0.0069	0.013
Animation	0.0018	0.0044	0.0021	0.003
Children	0.00014	0.00018	0.0021	2.8×10^{-5}
Comedy	**0.27**	0.2	0.082	0.24
Crime	0.0096	0.012	0.024	0.031
Documentary	0.021	0.014	0.012	0.082
Drama	0.14	0.11	0.03	0.055
Fantasy	0.037	0.034	0.007	0.024
Horror	0.13	0.16	0.12	**0.28**
IMAX	0	0	0	0
Musical	0.044	0.017	0.15	0.054
Mystery	0.0059	0.006	0.0063	0.01
Noir	0.096	0.0021	4.1×10^{-6}	0.021
Romance	0.034	0.12	**0.32**	0.034
Thriller	0.1	**0.22**	0.045	0.11
War	0.0062	0.0041	0.0018	0.0015
Western	0.074	0.045	0.14	0.02

Table 23: LLM genre counts at R@5 (2 SF), normalized column-wise. The most common genres selected per LLM are highlighted in bold.

Genre	BERT	RoBERTa Large	BART Large	ALBERT Large v2
Action	0.021	0.053	0.065	0.038
Adventure	0.026	0.027	0.013	0.017
Animation	0.0037	0.011	0.003	0.0088
Children	0.00021	0.00037	0.0021	5.6×10^{-5}
Comedy	**0.23**	**0.16**	0.091	0.19
Crime	0.015	0.018	0.034	0.047
Documentary	0.025	0.023	0.013	0.095
Drama	0.16	0.11	0.031	0.064
Fantasy	0.041	0.065	0.0099	0.041
Horror	0.094	0.14	0.093	**0.19**
IMAX	0	0	0	0
Musical	0.06	0.023	0.16	0.064
Mystery	0.013	0.018	0.012	0.019
Noir	0.09	0.0087	1.5×10^{-5}	0.044
Romance	0.06	0.13	**0.27**	0.045
Thriller	0.098	**0.16**	0.066	0.1
War	0.0071	0.0065	0.0021	0.0019
Western	0.061	0.056	0.14	0.037

Table 24: LLM genre counts at R@10 (2 SF), normalized column-wise. The most common genres selected per LLM are highlighted in bold.

	Ac	Adv	Ani	Chi	Com	Cri	Doc	Dra	Fan	Hor	IM	Mus	Mys	No	Rom	Thr	War	Wes
Ac	4	1	0	0	2	2	0	3	0	0	0	0	0	0	1	2	1	0
Adv	4	7	0	1	5	1	0	7	1	1	0	0	1	0	3	2	2	1
Ani	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Chi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Com	70	54	1	24	328	84	5	437	31	38	2	44	41	11	174	101	43	30
Cri	1	0	0	0	3	4	0	3	0	0	0	0	1	0	0	2	0	0
Doc	7	4	0	1	12	5	1	23	2	3	1	0	2	0	6	10	2	1
Dra	28	19	0	4	60	24	1	140	8	11	1	8	13	4	49	39	17	7
Fan	9	11	0	2	11	4	1	29	5	3	1	2	2	1	10	7	5	1
Hor	130	77	2	16	207	102	5	383	45	101	10	17	60	14	120	178	37	19
IM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mus	5	4	0	2	17	4	0	28	2	2	0	7	2	1	13	6	2	2
Mys	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
No	31	27	0	3	53	42	2	135	10	19	1	9	21	12	45	43	18	17
Rom	1	2	0	0	15	2	0	23	0	1	0	3	2	0	17	3	1	1
thr	28	13	0	2	41	26	1	84	5	10	2	3	13	3	28	46	7	3
war	4	1	0	0	2	1	0	8	0	0	0	0	0	0	2	2	5	1
wes	36	32	0	5	71	28	1	136	7	8	1	18	11	10	61	30	21	38
Sum	358	253	4	62	828	329	17	1439	117	197	19	113	172	58	530	469	161	119

Table 25: Error Matrix for BERT at R@1, averaged across all prompt styles, divided by the total number of prompt styles (81) to display the average genre counts for one prompt style for the whole filtered dataset. Rows represent the true genres, while columns represent the predicted genres. The diagonal cells display the true positives, while the non-diagonal cells represent the false positives (for the row genre) and false negatives (for the column genre).

	Ac	Adv	Ani	Chi	Com	Cri	Doc	Dra	Fan	Hor	IM	Mus	Mys	No	Rom	Thr	War	Wes
Ac	48	20	0	1	19	17	0	38	4	3	4	1	3	1	10	26	11	4
Adv	6	11	0	1	6	1	0	7	1	1	1	1	0	0	4	1	2	1
Ani	0	1	0	1	2	0	0	1	1	0	0	0	0	0	1	0	0	0
Chi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Com	142	109	5	55	725	163	9	794	58	51	3	104	65	19	357	169	59	52
Cri	4	1	0	0	8	11	0	14	0	1	0	0	3	1	2	6	1	1
Doc	6	4	0	1	19	6	2	41	2	3	0	1	3	1	13	11	4	1
Dra	16	12	0	2	32	14	1	101	5	5	1	3	7	2	28	23	12	5
Fan	24	32	0	5	15	5	1	42	19	3	5	3	4	0	13	11	6	1
Hor	139	80	1	19	233	125	5	475	69	216	10	19	91	21	135	255	40	27
IM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mus	2	2	0	2	21	3	0	28	1	2	0	16	1	0	19	3	1	2
Mys	1	1	0	0	2	2	0	3	0	0	0	0	2	0	1	2	0	0
No	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Rom	19	17	0	7	149	25	1	254	11	6	0	28	16	5	161	31	12	9
thr	317	170	3	28	375	270	15	1028	60	114	14	28	150	43	275	466	115	63
war	4	1	0	0	3	1	0	10	0	0	0	0	0	0	3	1	9	1
wes	34	33	0	3	57	20	1	120	4	5	0	14	8	6	51	21	21	51
Sum	763	495	10	128	1669	663	36	2955	237	409	38	218	354	100	1073	1026	294	218

Table 26: Error Matrix for RoBERTa at R@1, averaged across all prompt styles, divided by the total number of prompt styles (91) to display the average genre counts for one prompt style for the whole filtered dataset. Rows represent the true genres, while columns represent the predicted genres. The diagonal cells display the true positives, while the non-diagonal cells represent the false positives (for the row genre) and false negatives (for the column genre).

	Ac	Adv	Ani	Chi	Com	Cri	Doc	Dra	Fan	Hor	IM	Mus	Mys	No	Rom	Thr	War	Wes
Ac	18	7	0	1	8	8	0	11	1	1	1	0	1	0	2	12	2	1
Adv	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ani	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Chi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Com	4	4	0	2	34	6	0	25	2	2	0	4	2	1	13	5	2	2
Cri	1	1	0	0	2	3	0	3	0	0	0	0	1	1	1	2	0	0
Doc	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Dra	8	5	0	1	16	8	0	41	2	4	0	2	5	2	13	13	5	3
Fan	1	2	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Hor	27	14	0	3	38	22	1	74	12	49	2	3	18	3	19	55	6	3
IM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mus	1	1	0	1	7	1	0	9	0	1	0	7	0	0	6	1	0	1
Mys	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
No	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rom	1	1	0	0	15	2	0	19	1	1	0	3	1	0	16	2	1	0
thr	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
war	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
wes	5	4	0	0	5	2	0	12	0	0	0	1	1	1	5	2	2	12
Sum	67	40	1	9	129	53	2	200	22	57	4	20	30	7	77	94	20	22

Table 27: Error Matrix for BART at R@1, averaged across all prompt styles, divided by the total number of prompt styles (91) to display the average genre counts for one prompt style for the whole filtered dataset. Rows represent the true genres, while columns represent the predicted genres. The diagonal cells display the true positives, while the non-diagonal cells represent the false positives (for the row genre) and false negatives (for the column genre).

	Ac	Adv	Ani	Chi	Com	Cri	Doc	Dra	Fan	Hor	IM	Mus	Mys	No	Rom	Thr	War	Wes
Ac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Adv	1	2	0	0	2	0	0	2	0	0	0	0	0	0	1	0	0	0
Ani	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Chi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Com	42	28	0	10	158	49	2	217	15	21	1	23	23	7	89	65	17	16
Cri	2	0	0	0	4	6	0	5	0	0	0	0	1	0	1	2	0	0
Doc	13	8	0	2	25	10	1	41	4	7	1	3	5	1	14	18	4	4
Dra	7	5	0	1	16	7	0	35	2	4	0	2	4	2	12	10	5	3
Fan	1	2	0	0	1	0	0	3	0	1	0	0	0	0	1	1	0	0
Hor	118	77	1	16	186	94	5	364	36	74	8	21	53	15	120	158	44	28
IM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mus	4	3	0	2	14	4	0	25	2	3	0	5	2	1	10	6	2	2
Mys	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
No	0	0	0	0	1	1	0	3	0	0	0	0	1	0	1	1	0	0
Rom	1	1	0	0	12	2	0	15	1	0	0	2	1	0	13	2	1	0
thr	7	3	0	0	6	4	0	12	2	2	1	0	2	0	4	8	1	0
war	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
wes	1	1	0	0	1	0	0	2	0	0	0	0	0	0	1	0	0	1
Sum	200	132	3	32	425	177	8	725	63	114	12	56	93	27	267	272	77	56

Table 28: Error Matrix for ALBERT at R@1, averaged across all prompt styles, divided by the total number of prompt styles (91) to display the average genre counts for one prompt style for the whole filtered dataset. Rows represent the true genres, while columns represent the predicted genres. The diagonal cells display the true positives, while the non-diagonal cells represent the false positives (for the row genre) and false negatives (for the column genre).

Predicted Word	**Prediction Probability (3 SF)**
genre	0.304
comedy	0.0933
horror	0.0854
sgt	0.0686
film	0.0571
western	0.0295
vs	0.0293
noir	0.0273
drama	0.0258
films	0.0255
vol	0.0227
variety	0.0215
thriller	0.0176
itself	0.00971
worldwide	0.00784
dating	0.00714
again	0.00711
trilogy	0.00624
cinema	0.00612
fantasy	0.00588
musical	0.00572
pop	0.00499
documentary	0.00464
short	0.00447
romance	0.00430
british	0.00398
american	0.00332
jazz	0.00323
rock	0.00309
historical	0.00302
french	0.00299
television	0.00256
mainstream	0.00246
blues	0.00240
dance	0.00236
italian	0.00230
visual	0.00227
fiction	0.00210
adventure	0.00196
bollywood	0.00180
of	0.00162
adult	0.00160
german	0.00158
as	0.00151
war	0.00147
tango	0.00147
batman	0.00145
opera	0.00141
narrative	0.00140
classical	0.00140

Table 29: BERT's top 50 predicted words and their chances of being predicted at R@1.

Predicted Word	**Prediction Probability (3 SF)**
genre	0.205
thriller	0.198
comedy	0.177
horror	0.111
romance	0.0465
film	0.0324
western	0.0258
drama	0.0167
action	0.0100
fantasy	0.00975
variety	0.00769
documentary	0.00761
dating	0.00670
of	0.00665
cinema	0.00634
musical	0.00633
classic	0.00486
franchise	0.00378
family	0.00367
political	0.00346
french	0.00338
and	0.00316
series	0.00313
american	0.00303
crime	0.00282
british	0.00275
same	0.00266
category	0.00259
superhero	0.00245
films	0.00241
adventure	0.00239
new	0.00235
historical	0.00226
psychological	0.00184
war	0.00164
spy	0.00146
advertisement	0.00138
german	0.00130
christmas	0.00126
rock	0.00126
italian	0.00115
opera	0.00105
silent	0.00104
tamil	0.00101
wwii	0.000859
folk	0.000842
jazz	0.000830
aliens	0.000804
mystery	0.000670
short	0.000638

Table 30: RoBERTa's top 50 predicted words and their chances of being predicted at R@1.

Predicted Word	**Prediction Probability (3 SF)**
of	0.277
was	0.224
and	0.152
is	0.0913
american	0.0370
horror	0.0195
classic	0.0191
film	0.0183
world	0.0134
british	0.0117
french	0.00883
lost	0.00849
drama	0.00762
italian	0.00752
realm	0.00708
comedy	0.00664
growing	0.00612
german	0.00553
low	0.00380
romance	0.00372
dark	0.00348
action	0.00343
same	0.00321
western	0.00292
popularity	0.00231
musical	0.00204
indian	0.00202
mexican	0.00188
history	0.00165
japanese	0.00164
starring	0.00160
canadian	0.00156
dr	0.00141
science	0.00137
female	0.00115
popular	0.00107
swedish	0.00105
directed	0.00105
spanish	0.000904
australian	0.000885
russian	0.000820
crime	0.000812
tamil	0.000782
genre	0.000770
silent	0.000753
chinese	0.000648
story	0.000642
hong	0.000631
danish	0.000580
role	0.000539

Table 31: BART's top 50 predicted words and their chances of being predicted at R@1.

Predicted Word	**Prediction Probability (3 SF)**
genre	0.246
movie	0.123
horror	0.0790
studios	0.0571
comedy	0.0468
film	0.0414
cinema	0.0400
theaters	0.0278
francaise	0.0221
telenovela	0.0152
trilogy	0.0105
anime	0.0104
films	0.0102
documentary	0.00905
movies	0.00862
category	0.00801
theatre	0.00700
drama	0.00696
historical	0.00695
cinematic	0.00628
theater	0.00627
literary	0.00533
musical	0.00507
dating	0.00499
publishers	0.00473
opera	0.00469
american	0.00469
archives	0.00438
flick	0.00391
version	0.00374
bollywood	0.00368
picture	0.00334
revival	0.00307
romance	0.00297
italian	0.00294
canadian	0.00288
telugu	0.00287
thriller	0.00270
british	0.00263
french	0.00255
gallery	0.00243
tamil	0.00236
exists	0.00217
silent	0.00214
name	0.00209
group	0.00208
gangster	0.00202
german	0.00187
states	0.00186
malayalam	0.00180

Table 32: ALBERT's top 50 predicted words and their chances of being predicted at R@1.

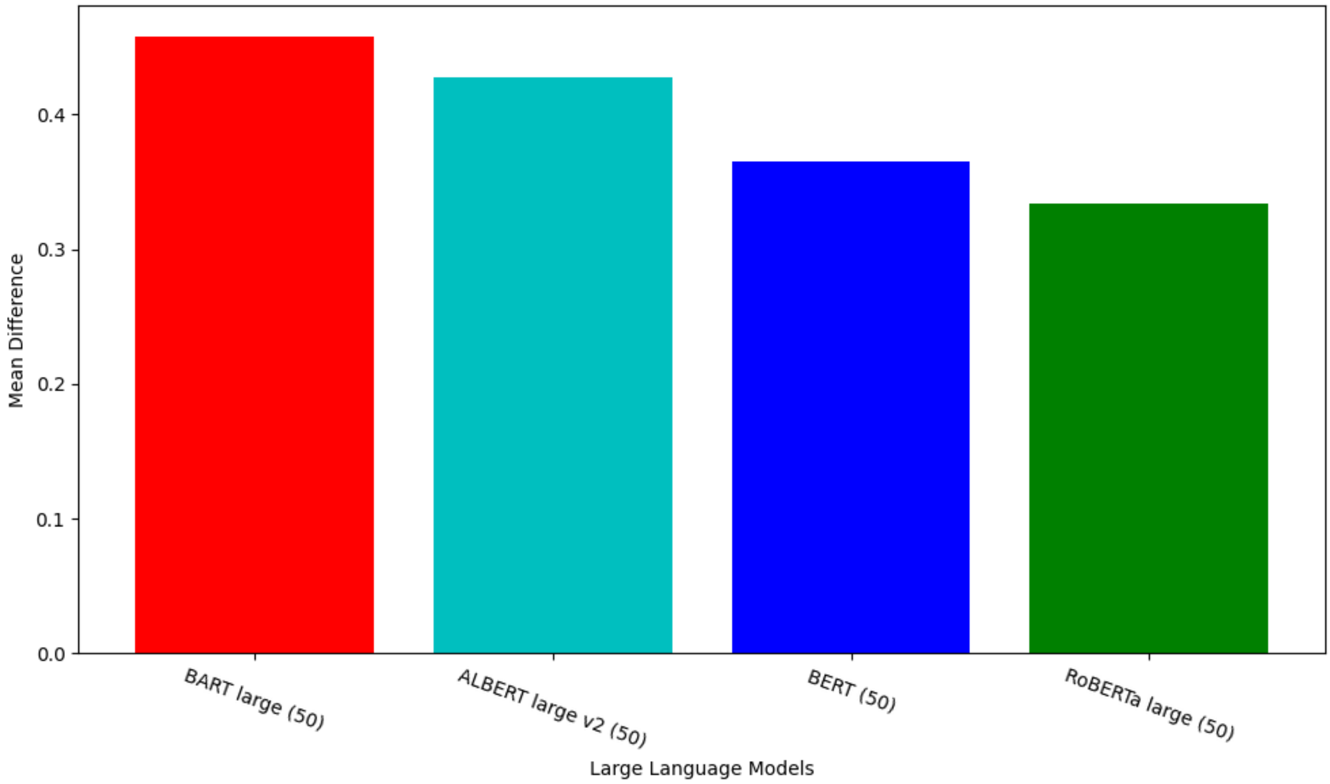

Figure 16: Mean Differences of the best performing prompts (in brackets) compared to the unenriched prompt 0 for each LLM at R@1.

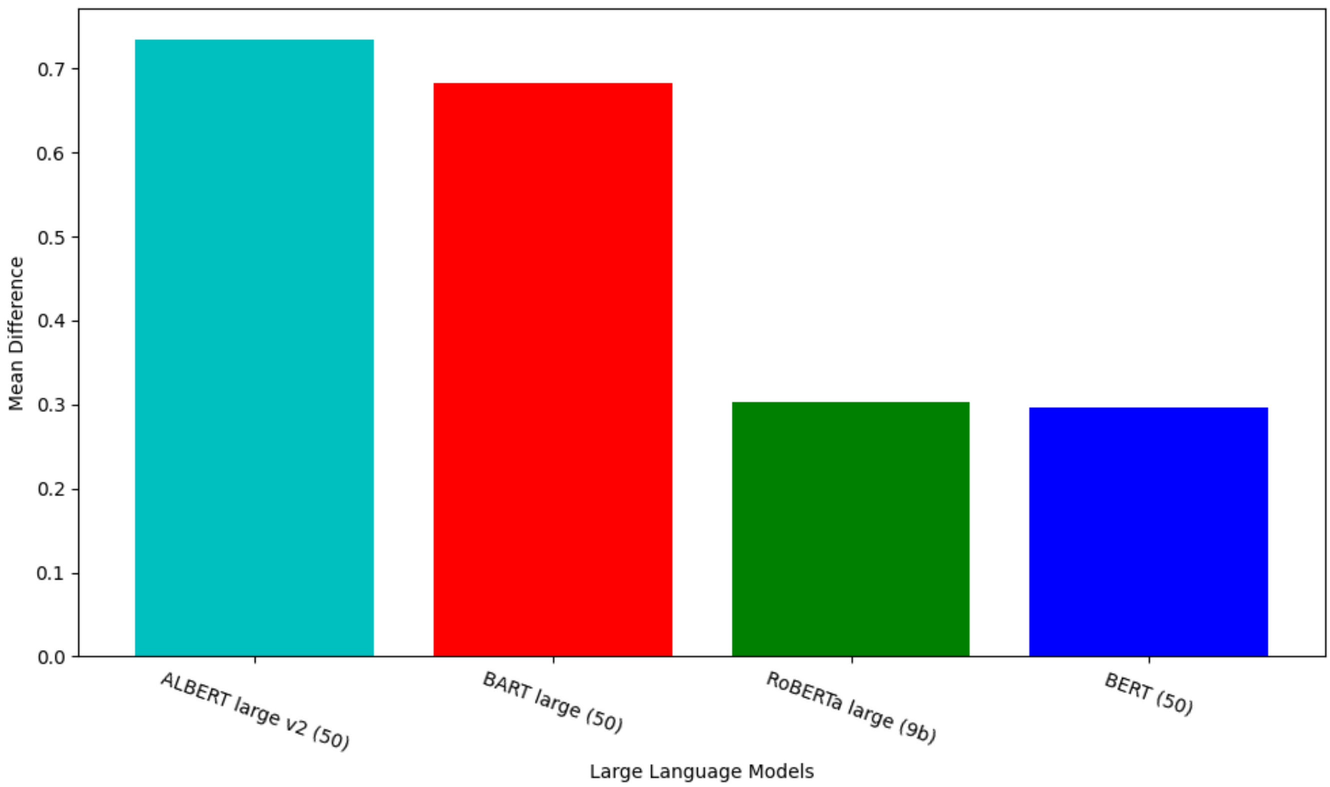

Figure 17: Mean Differences of the best performing prompts (in brackets) compared to the unenriched prompt 0 for each LLM at R@10.

B Appendix: Source Code

Contents of Source Code:

B.1 fetch_movies_kg.py

```python
# fetch_movies_kg.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Retrieves a dataset of movies knowledge graph properties from Wikidata.
#

# Imports.
from pandas import read_csv, merge
from SPARQLWrapper import SPARQLWrapper, JSON
from numpy import nan
from concurrent.futures import ThreadPoolExecutor, as_completed
from time import sleep
from pathlib import Path

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"

def merge_datasets():
    """Merges movies with their identifiers."""
    movies_df = read_csv(f"{PARENT_DIR}/Dataset/movies.csv")
    id_df = read_csv(f"{PARENT_DIR}/Dataset/links.csv")
    merged = merge(movies_df, id_df, on="movieId")
    merged.to_csv(f"{PARENT_DIR}/Dataset/movies_linked.csv", index=False)

def get_movie_data(imdbId, tmdbId):
    """Retrieves movie details from Wikidata using SPARQL endpoint.

    Details include cast, director, producer, screenwriter, composer,
    editor, distributor, and country. If the request fails, retries until
    success or timeout.

    Args:
        imdbId (str): IMDB identifier for the movie.
        tmdbId (str): TMDb identifier for the movie.

    Returns:
        dict: Dictionary containing the retrieved movie details.
    """
    # Declares global variables to be used across multiple concurrent
      threads.
    global movie_index, error_list
    # Wait 1 second every call to avoid server timeouts.
    sleep(1)
    movie_index += 1

    while True:
        print("Movie number:", movie_index)
        try:
            sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
            query = f"""
```

```
                SELECT
                    ?castLabel ?directorLabel ?producerLabel ?screenwriterLabel ?composerLabel ?editorLabel ?year ?distributorLabel ?countryLabel
                WHERE {{
                    {{ ?film wdt:P31 wd:Q11424 ; wdt:P345 "{int(imdbId)}" . }}
                    UNION
                    {{ ?film wdt:P31 wd:Q11424 ; wdt:P4947 "{int(tmdbId)}" . }}
                    OPTIONAL {{ ?film wdt:P161 ?cast . }}
                    OPTIONAL {{ ?film wdt:P57 ?director . }}
                    OPTIONAL {{ ?film wdt:P162 ?producer . }}
                    OPTIONAL {{ ?film wdt:P58 ?screenwriter . }}
                    OPTIONAL {{ ?film wdt:P86 ?composer . }}
                    OPTIONAL {{ ?film wdt:P1040 ?editor . }}
                    OPTIONAL {{ ?film wdt:P577 ?year . }}
                    OPTIONAL {{ ?film wdt:P750 ?distributor . }}
                    OPTIONAL {{ ?film wdt:P495 ?country . }}
                    SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }}
                }}
            """
            sparql.setQuery(query)
            sparql.setReturnFormat(JSON)
            results = sparql.query().convert()
            return results["results"]["bindings"]

        except Exception as e:
            error_list.append(str(movie_index) + str(e))
            # Wait 5 seconds to avoid server timeouts.
            sleep(5)
            # If we get timed out, do not retry the request.
            if str(e) != "HTTP Error 429: Too Many Requests":
                return None

def update_df(idx):
    """Updates DataFrame with the retrieved movie knowledge graph properties.

    Args:
        idx (int): Index of the row in the DataFrame to be updated.
    """
    # Declares global DataFrame to be used across multiple concurrent threads.
    global df
    data = get_movie_data(df.loc[idx, "imdbId"], df.loc[idx, "tmdbId"])
    if data:  # If data retrieval was successful.
        for item in data:  # For each Wikidata property.
            # Wikidata property labels.
            labels = ["castLabel", "directorLabel", "producerLabel",
                      "screenwriterLabel", "composerLabel", "editorLabel",
                      "distributorLabel", "countryLabel"]
```

```
            # Updates the respective columns in the DataFrame.
            for label in labels:
                column_name = label.replace("Label", "").lower()
                if label in item:
                    df.loc[idx, column_name] = item[label]["value"]

def fetch_knowledge_graph():
    """Uses multiple concurrent threads to fetch movie Wikidata properties."""
    # Declares global variables to be used across multiple concurrent threads.
    global df, movie_index, error_list
    movie_index = 0
    error_list = []

    df = read_csv(f"{PARENT_DIR}/Dataset/movies_linked.csv")
    # Drops rows with no genres.
    df["genres"] = df["genres"].replace({"(no genres listed)": nan})
    df = df.dropna(subset=["genres"])
    # Separates movie title and year into 2 columns.
    df["year"] = df["title"].apply(lambda x: x[-5:-1])
    df["title"] = df["title"].apply(lambda x: x[:-7])

    # 4 Multiple threads for improved runtime.
    completed_tasks = 0
    with ThreadPoolExecutor(max_workers=4) as executor:  # For each worker.
        # Attempts to update that workers row with Wikidata properties.
        futures = {executor.submit(update_df, idx): idx for idx in df.index}
        for _ in as_completed(futures):  # For each completed task.
            completed_tasks += 1

    # Outputs the full dataset to a CSV file.
    df.to_csv(f"{PARENT_DIR}/Dataset/movies_kg_full.csv", index=False)
    # Logs any encountered errors in a text file.
    with open(f"{PARENT_DIR}/Dataset/wikidata_error_file.txt", "w") as file:
        for item in error_list:
            file.write(str(item) + "\n")


def main():
    merge_datasets()
    fetch_knowledge_graph()

if __name__ == "__main__":
    main()
```

B.2 clean_movies_kg.py

```python
# clean_movies_kg.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Cleans a dataset of movie's knowledge graph properties.
#

# Imports.
from pandas import read_csv, DataFrame
from re import match
from pathlib import Path

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"

def remove_special_char_words(s):
    """Removes words containing special characters from a string.

    Args:
        s (str): Input string.

    Returns:
        str: Updated string with words containing special characters removed.
    """
    if isinstance(s, str):
        words = s.split()
        words = [word for word in words if match("^[a-zA-Z0-9_.\-|']*$", word)]
        s = " ".join(words)
    return s

def process_dataset():
    """Cleans and stores movies dataset and genre details."""
    # Loads the dataset.
    df = read_csv(f"{PARENT_DIR}/Dataset/movies_kg_full.csv")

    # Cleans the dataset.
    # Removes words with special characters.
    df = df.applymap(remove_special_char_words)
    # Lower cases all genres.
    df["genres"] = df["genres"].str.lower()
    # Replaces "film-noir" with "noir".
    df["genres"] = df["genres"].str.replace("film-noir", "noir")
    # Removes the sci-fi genres.
    df["genres"] = df["genres"].apply(lambda x: "|".join(
        [genre for genre in x.split("|") if genre != "sci-fi"]))
    # Removes all rows with empty genre values.
    df = df[df["genres"].str.strip() != ""]
    # Gets columns with string data type.
    string_columns = df.select_dtypes(include=[object]).columns.tolist()
```

```
    # Removes rows with whitespaces only.
    df = df[~df[string_columns].apply(lambda series:
        series.str.contains(r"^ *$", na=False)).any(axis=1)]
    # Removes rows with Wikidata identifiers instead of movie properties.
    df = df[~df[string_columns].apply(lambda series:
        series.str.contains(r"Q\d+", na=False)).any(axis=1)]
    # Drops any rows with missing values.
    df = df.dropna(how="any").reset_index(drop=True)

    # Counts genres.
    genres_df = df["genres"].str.get_dummies("|")
    genre_counts = genres_df.sum()
    genre_counts_df = DataFrame({"Genre": genre_counts.index,
                                 "Count": genre_counts.values})
    genre_counts_df["Percentage"] = (genre_counts_df["Count"] /
                                     df.shape[0]) * 100
    # Saves genre count and list of unique genres to CSV files.
    genre_counts_df.to_csv(f"{PARENT_DIR}/Dataset/genre_counts.csv",
                           index=False)
    unique_genres_df = DataFrame({"Unique_Genres": genre_counts.index})
    unique_genres_df.to_csv(f"{PARENT_DIR}/Dataset/unique_genres.csv",
                            index=False)

    # Removes useless IMDB and TMDB identifier columns.
    columns_out = df.columns.to_list()
    columns_out.remove("imdbId")
    columns_out.remove("tmdbId")
    # Saves cleaned dataset to CSV file.
    df.to_csv(f"{PARENT_DIR}/Dataset/movies_kg_cleaned.csv",
              columns=columns_out, index=False)

def main():
    process_dataset()

if __name__ == "__main__":
    main()
```

B.3 generate_prompts.py

```python
# generate_prompts.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Generates 91 prompts per movie. The generation of these prompts has been
    left
# computationally inefficient so that they are easier to read/understand.
#

# Imports.
from pandas import read_csv, concat
from time import time
from re import sub, escape, IGNORECASE
from numpy import nan
from transformers import (MarianMTModel, MarianTokenizer,
                          T5Tokenizer, T5ForConditionalGeneration)
from torch.utils.data import DataLoader
from torch.cuda import is_available, empty_cache
from torch import device, no_grad
from pathlib import Path
from glob import glob

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"
DEVICE = device("cuda" if is_available() else "cpu")
TRANS_BATCH_SIZE = 64
PARA_BATCH_SIZE = 256

def original(df):
    """Generates original prompts.

    Args:
        df (DataFrame): Filtered and cleaned movies dataset from which to
            construct the prompts.
    """
    prompts_df = df[["movieId", "title", "genres"]].copy()

    # Creates unenriched original prompts.
    prompts_df["0a"] = df["title"].apply(lambda x: f"{x} is a movie of the
        genre [MASK].")
    prompts_df["0c"] = df["title"].apply(lambda x: f"{x} is a movie of the
        genre <mask>.")

    # Creates enriched original prompts with KG properties separated by
       commas.
    prompts_df["1a"] = df.apply(lambda x: f"{x.title} is a movie, starring
        {x.cast}, of the genre [MASK].", axis=1)
    prompts_df["2a"] = df.apply(lambda x: f"{x.title} is a movie, starring
        {x.cast}, directed by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["3a"] = df.apply(lambda x: f"{x.title} is a movie, starring
        {x.cast}, directed by {x.director}, produced by {x.producer}, of
```

```
the genre [MASK].", axis=1)
    prompts_df["4a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, of the genre [MASK].", axis=1)
    prompts_df["5a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, music by {x.composer}, of the genre [MASK].", axis=1)
    prompts_df["6a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, music by {x.composer}, edited by {x.editor}, of the genre [MASK].", axis=1)
    prompts_df["7a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, music by {x.composer}, edited by {x.editor}, released {x.year}, of the genre [MASK].", axis=1)
    prompts_df["8a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, music by {x.composer}, edited by {x.editor}, released {x.year}, distributed by {x.distributor}, of the genre [MASK].", axis=1)
    prompts_df["9a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, produced by {x.producer}, screenwriter {x.screenwriter}, music by {x.composer}, edited by {x.editor}, released {x.year}, distributed by {x.distributor}, originating from {x.country}, of the genre [MASK].", axis=1)

    # Creates enriched original prompts with KG properties separated by the word "and".
    prompts_df["1b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and is of the genre [MASK].", axis=1)
    prompts_df["2b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and is of the genre [MASK].", axis=1)
    prompts_df["3b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and is of the genre [MASK].", axis=1)
    prompts_df["4b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and is of the genre [MASK].", axis=1)
    prompts_df["5b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and is of the genre [MASK].", axis=1)
    prompts_df["6b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and is of the genre [MASK].", axis=1)
    prompts_df["7b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and released {x.year} and is of the genre [
```

```
MASK].", axis=1)
    prompts_df["8b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and released {x.year} and distributed by {x.distributor} and is of the genre [MASK].", axis=1)
    prompts_df["9b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and released {x.year} and distributed by {x.distributor} and originating from {x.country} and is of the genre [MASK].", axis=1)

    # Creates another set of prompts with "<mask>" token instead.
    for i in range(1, 10):
        col_name_a = str(i) + "a"
        col_name_b = str(i) + "b"
        col_name_c = str(i) + "c"
        col_name_d = str(i) + "d"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)
        prompts_df[col_name_d] = prompts_df[col_name_b].replace(
            "\[MASK\]", "<mask>", regex=True)

    # Saves original prompts to CSV file.
    prompts_df.to_csv(f"{PARENT_DIR}/Prompts/original.csv", index=False)

def intermediate(df):
    """Generates intermediate prompts.

    Args:
        df (DataFrame): Filtered and cleaned movies dataset from which to
            construct the prompts.
    """
    prompts_df = df[["movieId", "title", "genres"]].copy()

    # Creates intermediate prompts with KG properties separated by commas.
    prompts_df["cast_a"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast}, of the genre [MASK].", axis=1)
    prompts_df["director_a"] = df.apply(lambda x: f"{x.title} is a movie directed by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["producer_a"] = df.apply(lambda x: f"{x.title} is a movie produced by {x.producer}, of the genre [MASK].", axis=1)
    prompts_df["screenwriter_a"] = df.apply(lambda x: f"{x.title} is a movie screenwriter {x.screenwriter}, of the genre [MASK].", axis=1)
    prompts_df["composer_a"] = df.apply(lambda x: f"{x.title} is a movie music by {x.composer}, of the genre [MASK].", axis=1)
    prompts_df["editor_a"] = df.apply(lambda x: f"{x.title} is a movie edited by {x.editor}, of the genre [MASK].", axis=1)
    prompts_df["year_a"] = df.apply(lambda x: f"{x.title} is a movie released {x.year}, of the genre [MASK].", axis=1)
    prompts_df["distributor_a"] = df.apply(lambda x: f"{x.title} is a movie distributed by {x.distributor}, of the genre [MASK].", axis
```

```
=1)
    prompts_df["country_a"] = df.apply(lambda x: f"{x.title} is a movie originating from {x.country}, of the genre [MASK].", axis=1)

    # Creates intermediate prompts with KG properties separated by the word "and".
    prompts_df["cast_b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and of the genre [MASK].", axis=1)
    prompts_df["director_b"] = df.apply(lambda x: f"{x.title} is a movie directed by {x.director} and of the genre [MASK].", axis=1)
    prompts_df["producer_b"] = df.apply(lambda x: f"{x.title} is a movie produced by {x.producer} and of the genre [MASK].", axis=1)
    prompts_df["screenwriter_b"] = df.apply(lambda x: f"{x.title} is a movie screenwriter {x.screenwriter} and of the genre [MASK].", axis=1)
    prompts_df["composer_b"] = df.apply(lambda x: f"{x.title} is a movie music by {x.composer} and of the genre [MASK].", axis=1)
    prompts_df["editor_b"] = df.apply(lambda x: f"{x.title} is a movie edited by {x.editor} and of the genre [MASK].", axis=1)
    prompts_df["year_b"] = df.apply(lambda x: f"{x.title} is a movie released {x.year} and of the genre [MASK].", axis=1)
    prompts_df["distributor_b"] = df.apply(lambda x: f"{x.title} is a movie distributed by {x.distributor} and of the genre [MASK].", axis=1)
    prompts_df["country_b"] = df.apply(lambda x: f"{x.title} is a movie originating from {x.country} and of the genre [MASK].", axis=1)

    # Creates another set of prompts with "<mask>" token instead.
    for i in ["cast", "director", "producer", "screenwriter", "composer",
              "editor", "year", "distributor", "country"]:
        col_name_a = str(i) + "_a"
        col_name_b = str(i) + "_b"
        col_name_c = str(i) + "_c"
        col_name_d = str(i) + "_d"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)
        prompts_df[col_name_d] = prompts_df[col_name_b].replace(
            "\[MASK\]", "<mask>", regex=True)

    # Saves intermediate prompts to CSV file.
    prompts_df.to_csv(f"{PARENT_DIR}/Prompts/intermediate.csv", index=False)

def custom(df):
    """Generates custom prompts.

    Args:
        df (DataFrame): Filtered and cleaned movies dataset from which to
            construct the prompts.
    """
    prompts_df = df[["movieId", "title", "genres"]].copy()

    # Creates custom prompts with KG properties separated by commas.
```

```
    prompts_df["10a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, of the genre [MASK].", axis=1)
    prompts_df["11a"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["12a"] = df.apply(lambda x: f"The movie {x.title} released in {x.year}, of the genre [MASK].", axis=1)
    prompts_df["13a"] = df.apply(lambda x: f"The movie {x.title} originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["14a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, directed by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["15a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, released in {x.year}, of the genre [MASK].", axis=1)
    prompts_df["16a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["17a"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director}, released in {x.year}, of the genre [MASK].", axis=1)
    prompts_df["18a"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["19a"] = df.apply(lambda x: f"The movie {x.title} released in {x.year}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["20a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, directed by {x.director}, released in {x.year}, of the genre [MASK].", axis=1)
    prompts_df["21a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, directed by {x.director}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["22a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, released in {x.year}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["23a"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director}, released in {x.year}, originating from {x.country}, of the genre [MASK].", axis=1)
    prompts_df["24a"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast}, directed by {x.director}, released in {x.year}, originating from {x.country}, of the genre [MASK].", axis=1)

    # Creates custom prompts with KG properties separated by the word "and".
    prompts_df["10b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and of the genre [MASK].", axis=1)
    prompts_df["11b"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director} and of the genre [MASK].", axis=1)
    prompts_df["12b"] = df.apply(lambda x: f"The movie {x.title} released in {x.year} and of the genre [MASK].", axis=1)
    prompts_df["13b"] = df.apply(lambda x: f"The movie {x.title} originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["14b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and directed by {x.director} of the genre [MASK].", axis=1)
```

```python
    prompts_df["15b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and released in {x.year} and of the genre [MASK].", axis=1)
    prompts_df["16b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["17b"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director} and released in {x.year} and of the genre [MASK].", axis=1)
    prompts_df["18b"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["19b"] = df.apply(lambda x: f"The movie {x.title} released in {x.year} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["20b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and directed by {x.director} and released in {x.year} and of the genre [MASK].", axis=1)
    prompts_df["21b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and directed by {x.director} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["22b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and released in {x.year} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["23b"] = df.apply(lambda x: f"The movie {x.title} directed by {x.director} and released in {x.year} and originating from {x.country} and of the genre [MASK].", axis=1)
    prompts_df["24b"] = df.apply(lambda x: f"The movie {x.title} starring {x.cast} and directed by {x.director} and released in {x.year} and originating from {x.country} and of the genre [MASK].", axis=1)

    # Creates naturally worded custom prompts.
    prompts_df["25a"] = df.apply(lambda x: f"From the mind of {x.director} and brought to life by {x.cast}, {x.title} is a noteworthy addition to the [MASK] genre.", axis=1)
    prompts_df["26a"] = df.apply(lambda x: f"With {x.title}, {x.director} brings a new twist to the [MASK] genre, featuring powerful performances by {x.cast}.", axis=1)
    prompts_df["27a"] = df.apply(lambda x: f"The [MASK] genre is beautifully represented in {x.country} through the movie {x.title}, featuring the unique performance of {x.cast}.", axis=1)
    prompts_df["28a"] = df.apply(lambda x: f"Through the lens of {x.director}, {x.title} blends gripping performances by {x.cast} with the nuanced themes of the [MASK] genre.", axis=1)
    prompts_df["29a"] = df.apply(lambda x: f"{x.title} is a remarkable exploration of the [MASK] genre, driven by the stellar direction of {x.director} and compelling acting from {x.cast}.", axis=1)
    prompts_df["30a"] = df.apply(lambda x: f"Immersing audiences in the [MASK] genre, {x.director} creates a cinematic gem with {x.title}, featuring a standout performance by {x.cast}.", axis=1)
    prompts_df["31a"] = df.apply(lambda x: f"A film released in {x.year} from {x.country}, {x.title} features {x.cast} and falls into the [MASK] genre under the direction of {x.director}.", axis=1)
```

```python
    prompts_df["32a"] = df.apply(lambda x: f"{x.title}, a masterpiece in the [MASK] genre from {x.year}, reflects {x.director}\"s vision and {x.country}\"s culture, starring {x.cast}.", axis=1)
    prompts_df["33a"] = df.apply(lambda x: f"{x.director} crafts a vibrant narrative within the [MASK] genre in {x.year}\"s {x.title}, encapsulating the heartbeat of {x.country} with an unforgettable performance by {x.cast}.", axis=1)
    prompts_df["34a"] = df.apply(lambda x: f"{x.title}, a cinematic treat from {x.country} released in {x.year}, weaves a compelling [MASK] narrative under the mastery of {x.director}, featuring {x.cast}.", axis=1)
    prompts_df["35a"] = df.apply(lambda x: f"Under the masterful direction of {x.director}, {x.title} was released in {x.year}, representing the unique spirit of {x.country}\"s film industry, while also creating a fresh narrative in the [MASK] genre, featuring the remarkable talents of {x.cast}.", axis=1)
    prompts_df["36a"] = df.apply(lambda x: f"In {x.year}, the film world was enriched by {x.title}, a significant [MASK] genre movie hailing from {x.country}, guided by the innovative vision of director {x.director} and showcasing the notable performances of {x.cast}.", axis=1)

    # Creates another set of prompts with "<mask>" token instead.
    for i in range(10, 25):
        col_name_a = str(i) + "a"
        col_name_b = str(i) + "b"
        col_name_c = str(i) + "c"
        col_name_d = str(i) + "d"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)
        prompts_df[col_name_d] = prompts_df[col_name_b].replace(
            "\[MASK\]", "<mask>", regex=True)
    for i in range(25, 37):
        col_name_a = str(i) + "a"
        col_name_c = str(i) + "c"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)

    # Saves custom prompts to CSV file.
    prompts_df.to_csv(f"{PARENT_DIR}/Prompts/custom.csv", index=False)


def translate_batch(texts, tokenizer, model):
    """Translates a batch of texts without translating the [MASK] token.

    Args:
        texts (list of str): Texts to be translated.
        tokenizer: Tokenizer corresponding to the model.
        model: Pre-trained translation model.

    Returns:
        list of str: Translated texts.
    """
```

```
    # Brings the appropriate model to the GPU.
    model.to(DEVICE)
    # Adds a placeholder for the [MASK] token so that it does not get
    # translated.
    mask_placeholder = "#1  /?!"
    texts = [text.replace("[MASK]", mask_placeholder) for text in texts]
    # Tokenizes the input.
    inputs = tokenizer(texts, return_tensors="pt", padding=True).to(DEVICE)
    # Translates the batch of prompts.
    translated = model.generate(**inputs, max_length=1024)
    translated_texts = [
        tokenizer.decode(t, skip_special_tokens=True) for t in translated]
    # Readd the [MASK] token.
    translated_texts = [sub(escape(mask_placeholder), "[MASK]", text,
                            flags=IGNORECASE) for text in translated_texts]
    # Brings the appropriate model back to the CPU to avoid memory errors.
    model.to("cpu")
    return translated_texts

def round_trip_translate(texts, models, tokenizers):
    """Translates and back-translates texts.

    Args:
        texts (list of str): Texts to be translated.
        models (tuple): Source to target and target to source translation models.
        tokenizers (tuple): Source to target and target to source tokenizers.

    Returns:
        list of str: Back-translated texts.
    """
    # Translates the texts to the target language.
    translated_texts = translate_batch(texts, tokenizers[0], models[0])
    # Translates the texts back to English.
    back_translated_texts = translate_batch(translated_texts, tokenizers[1],
                                            models[1])
    return back_translated_texts

def translated(df):
    """Performs round trip translation on the best-performing prompts.

    Args:
        df (DataFrame): DataFrame containing movie details.
    """
    prompts_df = df[["movieId", "title", "genres"]].copy()

    # Regenerates the best-performing prompts.
    prompts_df["2a"] = df.apply(lambda x: f"{x.title} is a movie, starring {x.cast}, directed by {x.director}, of the genre [MASK].", axis=1)
```

```python
    prompts_df["7b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and released {x.year} and is of the genre [MASK].", axis=1)
    prompts_df["9b"] = df.apply(lambda x: f"{x.title} is a movie starring {x.cast} and directed by {x.director} and produced by {x.producer} and screenwriter {x.screenwriter} and music by {x.composer} and edited by {x.editor} and released {x.year} and distributed by {x.distributor} and originating from {x.country} and is of the genre [MASK].", axis=1)
    prompts_df["31a"] = df.apply(lambda x: f"A film released in {x.year} from {x.country}, {x.title} features {x.cast} and falls into the [MASK] genre under the direction of {x.director}.", axis=1)
    prompts_df["32a"] = df.apply(lambda x: f"{x.title}, a masterpiece in the [MASK] genre from {x.year}, reflects {x.director}\"s vision and {x.country}\"s culture, starring {x.cast}.", axis=1)

    languages = ["fr", "de"]
    prompt_number = 37
    for prompt in ["2a", "7b", "9b", "31a", "32a"]:  # For each of the prompts.
        for lang in languages:  # For each of French and German.
            batch_counter = 0
            col_name = str(prompt_number) + "a"
            print("Column:", col_name)
            prompt_number += 1
            texts = list(prompts_df[prompt].values)
            translated_list = []
            # For each batch.
            for batch in DataLoader(texts, batch_size=TRANS_BATCH_SIZE):
                batch_counter += TRANS_BATCH_SIZE
                print("Prompt:", batch_counter)

                # Defines models and tokenizers based on the target languages.
                models = {
                    "fr": (
                        MarianMTModel.from_pretrained(
                            "Helsinki-NLP/opus-mt-en-fr"),
                        MarianMTModel.from_pretrained(
                            "Helsinki-NLP/opus-mt-fr-en")
                    ),
                    "de": (
                        MarianMTModel.from_pretrained(
                            "Helsinki-NLP/opus-mt-en-de"),
                        MarianMTModel.from_pretrained(
                            "Helsinki-NLP/opus-mt-de-en")
                    )
                }
                tokenizers = {
                    "fr": (
                        MarianTokenizer.from_pretrained(
```

```
                    "Helsinki-NLP/opus-mt-en-fr"),
                MarianTokenizer.from_pretrained(
                    "Helsinki-NLP/opus-mt-fr-en")
            ),
            "de": (
                MarianTokenizer.from_pretrained(
                    "Helsinki-NLP/opus-mt-en-de"),
                MarianTokenizer.from_pretrained(
                    "Helsinki-NLP/opus-mt-de-en")
            )
        }

        start_time = time()

        try:
            # These specific batches crash CUDA, so skip them.
            if batch_counter in [4608, 5824]:
                # NAN values for these specific batches.
                back_translated_batch = [nan] * TRANS_BATCH_SIZE
            else:
                back_translated_batch = round_trip_translate(
                    batch, models[lang], tokenizers[lang])

        except Exception as e:  # This should not occur.
            # Print and output the error to a text file if there
               is
            # one.
            with open(f"{PARENT_DIR}/Prompts/Translated/
               error_batch.txt", "w") as file:
                file.write(str(e) + str(batch_counter))
            back_translated_batch = [nan] * TRANS_BATCH_SIZE

        translated_list.extend(back_translated_batch)

        # Attempts to clear any unused variables and the CUDA
           cache to
        # avoid CUDA crashing.
        del back_translated_batch, models, tokenizers
        try: empty_cache()
        except: pass

    prompts_df[col_name] = translated_list

    # Saves current translated prompt style to CSV file.
    prompts_df.to_csv(
        f"{PARENT_DIR}/Prompts/Translated/{col_name}.csv",
        columns=["movieId", "title", "genres"] + [col_name],
        index=False)
    with open(f"{PARENT_DIR}/Prompts/Translated/{col_name}_runtime
       .txt",
              "w") as file:
        file.write(str(time() - start_time))

```

```
            # Attempts to clear the CUDA cache to avoid CUDA crashing.
            try: empty_cache()
            except: pass


    all_files = glob(f"{PARENT_DIR}/Prompts/Translated/*.csv")
    files = []
    for filename in all_files:
        file_df = read_csv(filename, index_col=None)
        files.append(file_df)

    combined_df = concat(files, axis=0, ignore_index=True)
    # Combining entries using "movieId"
    combined_df = combined_df.groupby("movieId").first().reset_index()

    # Create another set of prompts with "<mask>" token
    for i in range(37, 47):
        col_name_a = str(i) + "a"
        col_name_c = str(i) + "c"
        combined_df[col_name_c] = combined_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)

    combined_df.to_csv(f"{PARENT_DIR}/Prompts/translated.csv", index=False)


def paraphrase_batch(texts, model, tokenizer):
    """Paraphrases a batch of texts using a given model and tokenizer.

    Args:
        texts (list): A list of texts to be paraphrased.
        model (T5ForConditionalGeneration): The T5 model used for paraphrasing.
        tokenizer (T5Tokenizer): The tokenizer for the T5 model.

    Returns:
        list: A list of paraphrased texts.
    """
    placeholder = "comedy"
    texts_with_ph = [text.replace("[MASK]", placeholder) for text in texts]
    input_ids = tokenizer.batch_encode_plus(
        ["paraphrase: " + t for t in texts_with_ph], return_tensors="pt",
        padding=True, truncation=True, max_length=512)["input_ids"].to(DEVICE)

    with no_grad():
        outputs = model.generate(input_ids, max_length=100,
                                 num_return_sequences=1)

    return [tokenizer.decode(output, skip_special_tokens=True).replace(
        placeholder, "[MASK]") for output in outputs]

```

```python
def paraphrase(df):
    """Paraphrases prompts using the Flan-T5 LLM.

    Args:
        df (DataFrame): DataFrame containing movie data.
    """
    # Loads model and tokenizer.
    model_name = "google/flan-t5-base"
    tokenizer = T5Tokenizer.from_pretrained(model_name)
    model = T5ForConditionalGeneration.from_pretrained(model_name).to(
        DEVICE)

    # Regenerates best-performing prompts.
    prompts_df = df[["movieId", "title", "genres"]].copy()
    prompts_df["2a"] = df.apply(lambda x: f"{x.title} is a movie, starring
         {x.cast}, directed by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["7b"] = df.apply(lambda x: f"{x.title} is a movie starring
        {x.cast} and directed by {x.director} and produced by {x.producer}
        and screenwriter {x.screenwriter} and music by {x.composer} and
        edited by {x.editor} and released {x.year} and is of the genre [
        MASK].", axis=1)
    prompts_df["9b"] = df.apply(lambda x: f"{x.title} is a movie starring
        {x.cast} and directed by {x.director} and produced by {x.producer}
        and screenwriter {x.screenwriter} and music by {x.composer} and
        edited by {x.editor} and released {x.year} and distributed by {x.
        distributor} and originating from {x.country} and is of the genre [
        MASK].", axis=1)
    prompts_df["31a"] = df.apply(lambda x: f"A film released in {x.year}
        from {x.country}, {x.title} features {x.cast} and falls into the [
        MASK] genre under the direction of {x.director}.", axis=1)
    prompts_df["32a"] = df.apply(lambda x: f"{x.title}, a masterpiece in
        the [MASK] genre from {x.year}, reflects {x.director}\"s vision and
         {x.country}\"s culture, starring {x.cast}.", axis=1)

    prompt_number = 47
    for prompt_key in ["2a", "7b", "9b", "31a", "32a"]:  # For each prompt
        .
        all_prompts = list(prompts_df[prompt_key])
        paraphrased_batch = []

        # Paraphrases prompts batch by batch.
        for i in range(0, len(all_prompts), PARA_BATCH_SIZE):
            batch = all_prompts[i : i + PARA_BATCH_SIZE]
            paraphrased_batch.extend(paraphrase_batch(batch, model,
                tokenizer))

        # Adds paraphrased prompts to DataFrame.
        prompts_df[str(prompt_number) + "a"] = paraphrased_batch
        prompt_number += 1

    # Creates another set of prompts with "<mask>" token instead.
    for i in range(47, 52):
        col_name_a = str(i) + "a"
```

```python
        col_name_c = str(i) + "c"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)

    # Saves the DataFrame to a CSV file.
    prompts_df.to_csv(f"{PARENT_DIR}/Prompts/paraphrased.csv", index=False)


def thesaurus(df):
    prompts_df = df[["movieId", "title", "genres"]].copy()

    # Creates 3 thesaurus-paraphrased prompts for each best-performing prompt.
    prompts_df["52a"] = df.apply(lambda x: f"{x.title} is a film, featuring {x.cast}, controlled by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["53a"] = df.apply(lambda x: f"{x.title} is a flick, performing {x.cast}, supervised by {x.director}, of the genre [MASK].", axis=1)
    prompts_df["54a"] = df.apply(lambda x: f"{x.title} is a picture, appearing {x.cast}, guided by {x.director}, of the genre [MASK].", axis=1)

    prompts_df["55a"] = df.apply(lambda x: f"{x.title} is a film featuring {x.cast} and controlled by {x.director} and made by {x.producer} and scriptwriter {x.screenwriter} and melody by {x.composer} and corrected by {x.editor} and announced {x.year} and is of the genre [MASK].", axis=1)
    prompts_df["56a"] = df.apply(lambda x: f"{x.title} is a flick performing {x.cast} and supervised by {x.director} and created by {x.producer} and playwright {x.screenwriter} and harmony by {x.composer} and modified by {x.editor} and distributed {x.year} and is of the genre [MASK].", axis=1)
    prompts_df["57a"] = df.apply(lambda x: f"{x.title} is a picture appearing {x.cast} and guided by {x.director} and formed by {x.producer} and scripter {x.screenwriter} and tune by {x.composer} and revised by {x.editor} and issued {x.year} and is of the genre [MASK].", axis=1)

    prompts_df["58a"] = df.apply(lambda x: f"{x.title} is a film featuring {x.cast} and controlled by {x.director} and made by {x.producer} and scriptwriter {x.screenwriter} and melody by {x.composer} and corrected by {x.editor} and announced {x.year} and allocated by {x.distributor} and arising from {x.country} and is of the genre [MASK].", axis=1)
    prompts_df["59a"] = df.apply(lambda x: f"{x.title} is a flick performing {x.cast} and supervised by {x.director} and created by {x.producer} and playwright {x.screenwriter} and harmony by {x.composer} and modified by {x.editor} and distributed {x.year} and alloted by {x.distributor} and developing from {x.country} and is of the genre [MASK].", axis=1)
    prompts_df["60a"] = df.apply(lambda x: f"{x.title} is a picture
```

```
appearing {x.cast} and guided by {x.director} and formed by {x.producer} and scripter {x.screenwriter} and tune by {x.composer} and revised by {x.editor} and issued {x.year} and dispensed by {x.distributor} and growing from {x.country} and is of the genre [MASK].", axis=1)

    prompts_df["61a"] = df.apply(lambda x: f"A movie announced in {x.year} out of {x.country}, {x.title} shows {x.cast} and lapses into the [MASK] genre below the management of {x.director}.", axis=1)
    prompts_df["62a"] = df.apply(lambda x: f"A flick distributed in {x.year} arising out of {x.country}, {x.title} displays {x.cast} and drifts into the [MASK] genre beneath the administration of {x.director}.", axis=1)
    prompts_df["63a"] = df.apply(lambda x: f"A picture issued in {x.year} coming out of {x.country}, {x.title} exhibits {x.cast} and resorts the [MASK] genre underneath the government of {x.director}.", axis=1)

    prompts_df["64a"] = df.apply(lambda x: f"{x.title}, a masterwork in the [MASK] genre out of {x.year}, shows {x.director}\"s invention and {x.country}\"s lifestyle, featuring {x.cast}.", axis=1)
    prompts_df["65a"] = df.apply(lambda x: f"{x.title}, a coup in the [MASK] genre arising out of {x.year}, depicts {x.director}\"s creativity and {x.country}\"s customs, performing {x.cast}.", axis=1)
    prompts_df["66a"] = df.apply(lambda x: f"{x.title}, a classic in the [MASK] genre coming out of {x.year}, characterizes {x.director}\"s inventiveness and {x.country}\"s traditions, appearing {x.cast}.", axis=1)

    # Creates another set of prompts with "<mask>" token instead.
    for i in range(52, 67):
        col_name_a = str(i) + "a"
        col_name_c = str(i) + "c"
        prompts_df[col_name_c] = prompts_df[col_name_a].replace(
            "\[MASK\]", "<mask>", regex=True)

    # Saves prompts to CSV file.
    prompts_df.to_csv(f"{PARENT_DIR}/Prompts/thesaurus.csv", index=False)


def main():
    # Reads in filtered and cleaned movies dataset.
    df = read_csv(f"{PARENT_DIR}/Dataset/movies_kg_cleaned.csv")

    original(df)

    intermediate(df)

    custom(df)

    translated(df)

```

```
    paraphrase(df)

    thesaurus(df)

if __name__ == "__main__":
    main()
```

B.4 probe_llms.py

```
# probe_llms.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Probes a range of LLMs with the constructed prompts.
#

# Imports.
from pandas import read_csv
from time import time
from torch.cuda import is_available, empty_cache
from torch import device, no_grad, tensor, topk
from re import match
from tqdm import tqdm
from transformers import (BertTokenizer, BertForMaskedLM,
                          RobertaTokenizer, RobertaForMaskedLM,
                          BartTokenizer, BartForConditionalGeneration,
                          AlbertTokenizer, AlbertForMaskedLM)
from pathlib import Path

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"
DEVICE = device("cuda" if is_available() else "cpu")
INTERMEDIATE_COLUMNS = ["title", "genres",
                        "cast_a", "cast_b", "director_a", "director_b",
                        "producer_a", "producer_b", "screenwriter_a",
                        "screenwriter_b", "composer_a", "composer_b",
                        "editor_a", "editor_b", "year_a", "year_b",
                        "distributor_a", "distributor_b",
                        "country_a", "country_b"]

def probe(folder):
    """Probe 4 models with a given set of prompts.

    Args:
        folder (str): Prompt type to probe models with.
    """
    df = read_csv(f"{PARENT_DIR}/Prompts/{folder.lower()}.csv")

    # Defines models and their column IDs.
    models = [("bert-base-uncased", BertTokenizer, BertForMaskedLM, ["a",
       "b"]),
              ("roberta-large", RobertaTokenizer, RobertaForMaskedLM, ["c"
                 , "d"]),
              ("facebook/bart-large", BartTokenizer,
                 BartForConditionalGeneration, ["c", "d"]),
              ("albert-large-v2", AlbertTokenizer, AlbertForMaskedLM, ["a"
                 , "b"])]

    # For each model.
    for model_name, Tokenizer, Model, column_ids in tqdm(models, desc="
```

```
Models"):
        # Declares a fresh copy of the DataFrame each iteration.
        df_copy = df.copy()
        result_columns = ["title", "genres"]
        props = ["cast", "director", "producer", "screenwriter",
                 "composer", "editor", "year", "distributor", "country"]

        # Loads model and tokenizer.
        tokenizer = Tokenizer.from_pretrained(model_name)
        model = Model.from_pretrained(model_name)
        model.to(DEVICE)
        model.eval()

        # Select prompt columns based on the folder.
        if folder == "Original":
            prompt_columns = (["0" + column_ids[0]] +
                              [f"{i}{c}" for i in range(1, 10) for c in column_ids])
        elif folder == "Custom":
            prompt_columns = ([f"{i}{column_ids[0]}" for i in range(10, 37)] +
                              [f"{i}{column_ids[1]}" for i in range(10, 25)])
        elif folder == "Translated":
            prompt_columns = [f"{i}{column_ids[0]}" for i in range(37, 47)]
        elif folder == "Paraphrased":
            prompt_columns = [f"{i}{column_ids[0]}" for i in range(47, 52)]
        elif folder == "Thesaurus":
            prompt_columns = [f"{i}{column_ids[0]}" for i in range(52, 67)]
        elif folder == "Intermediate":
            prompt_columns = [f"{i}_{c}" for i in props for c in column_ids]
            result_columns.extend([f"{i}" for i in prompt_columns])

        start_time = time()
        # For each prompt.
        for column in tqdm(prompt_columns, desc="Prompts", leave=False):
            print("\nCurrent prompt style:", column)
            if folder != "Intermediate":
                # Standardize column names.
                if column == "0a" or column == "0c":
                    result_column_base = "0"
                else:
                    result_column_base = column.replace("c", "a").replace("d", "b")
            else:
                result_column_base = column
            result_columns.append(result_column_base)

            for i, prompt in enumerate(df_copy[column]):  # For each
```

```
               prompt.
                try:
                    # Tokenizes input..
                    tokens = tokenizer.encode(prompt, add_special_tokens=True)
                    # Moves tensor to GPU if available, otherwise CPU.
                    input_ids = tensor(tokens).unsqueeze(0).to(DEVICE)
                    # Calculates predicted tokens instead of the mask token.
                    with no_grad():
                        predictions = model(input_ids).logits[
                            0, tokens.index(tokenizer.mask_token_id)]
                    predicted_tokens = []
                    # For top 1000 predicted tokens.
                    for id in topk(predictions, 1000).indices:
                        # Gets predicted word, removes any whitespaces.
                        word = tokenizer.decode([id]).strip()
                        word = word.replace(" ", "").lower()
                        # If word does not contain special characters and
                        # is not empty and has not already been added.
                        if (match("^[a-zA-Z]*$", word) and word != "" and
                            word not in predicted_tokens):
                            predicted_tokens.append(word)
                            # If we have 10 words, breaks the for loop.
                            if len(predicted_tokens) == 10: break

                    # Saves predictions in DataFrame.
                    df_copy.at[
                        i, result_column_base] = "|".join(predicted_tokens)

                # If error, saves empty string as predicted words list.
                except Exception as e:
                    # print("Exception: " + str(e))
                    df_copy.at[i, result_column_base] = ""

        # Saves results to CSV file.
        result_columns = list(dict.fromkeys(result_columns))
        filename = f"{PARENT_DIR}/Predictions/{folder}/{model_name.split('/')[-1]}.csv"
        if folder != "Intermediate":
            df_copy[result_columns].to_csv(filename, index=False)
        else:
            df_copy.to_csv(filename, index=False, columns=INTERMEDIATE_COLUMNS)
        time_taken = time() - start_time

        # Saves the time taken for the current LLM in a separate text file.
        filename = f"{PARENT_DIR}/Predictions/{folder}/{model_name.split('/')[-1]}_runtime.txt"
        with open(filename, "w") as file:
            file.write(str(time_taken))
```

```

        # Attempts to clear any unused variables and the CUDA cache to
            avoid
        # CUDA crashing.
        del tokenizer, model
        try: empty_cache()
        except: pass

def main():
    probe("Original")
    probe("Intermediate")
    probe("Custom")
    probe("Translated")
    probe("Paraphrased")
    probe("Thesaurus")

if __name__ == "__main__":
    main()
```

B.5 stats_eval.py

```python
# stats_eval.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Statistically evaluates the movie prediction results.
#

# Imports.
from pandas import read_csv, concat, notnull, isna, DataFrame, MultiIndex
from pandas.errors import PerformanceWarning
from os import listdir
from collections import Counter
from numpy import nan
from warnings import filterwarnings
from pathlib import Path
import os

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"
TOTAL_STYLES = 67
RESULT_COLUMNS = (["0"] + [str(i) + "a" for i in range(1, 25)] +
                  [str(i) + "b" for i in range(1, 25)] +
                  [str(i) for i in range(25, TOTAL_STYLES)])
RECALLS = [1, 5, 10]
REPLACEMENTS = {"love": "romance", "romantic": "romance",
                "comedic": "comedy", "comedies": "comedy",
                "animated": "animation", "music": "musical"}
LLM_NAMES = ["bert-base-uncased", "roberta-large",
             "facebook/bart-large", "albert-large-v2"]

# Disables relevant warnings.
filterwarnings("ignore", category=PerformanceWarning)


def calculate_runtimes():
    """Calculates runtimes for all prompt styles."""
    total_sums = {}
    llm_filenames = ["bert-base-uncased", "roberta-large",
                     "bart-large", "albert-large-v2"]
    breakdown_sums = {llm: {} for llm in llm_filenames}
    # For each LLM.
    for llm in llm_filenames:
        total = 0
        # For each prompt style grouping.
        for directory_path in ["Original", "Intermediate", "Custom",
                               "Translated", "Paraphrased", "Thesaurus"]:
            dir_total = 0
            # For each file.
            for filename in listdir(f"{PARENT_DIR}/Predictions/{
                directory_path}"):
                # Checks if the file ends with the current LLM and if it's
```

```
                    a
                # text file.
                if filename.endswith(f"{llm}_runtime.txt"):
                    # Opens the file and adds its content to the total
                    with open(f"{PARENT_DIR}/Predictions/{directory_path}/{filename}", "r") as f:
                        runtime = float(f.read())
                        dir_total += runtime

            # Converts directory runtime from seconds to hours and minutes.
            hours, remainder = divmod(dir_total, 3600)
            minutes, _ = divmod(remainder, 60)
            breakdown_sums[llm][directory_path] = f"{int(hours)} hours {int(minutes)} minutes"

            total += dir_total

        # Converts the runtime from seconds to hours and minutes.
        hours, remainder = divmod(total, 3600)
        minutes, _ = divmod(remainder, 60)

        # Add the total for the current model to the dictionary
        total_sums[llm] = f"{int(hours)} hours {int(minutes)} minutes"

    rows = []
    for llm, runtime in total_sums.items():  # For each LLM and its runtime.
        row = {"LLM": llm, "Runtime": runtime}
        row.update(breakdown_sums[llm])
        rows.append(row)

    # Exports the runtimes to a CSV file.
    df = DataFrame(rows)
    df.to_csv(f"{PARENT_DIR}/Results/Summaries/runtimes.csv", index=False)

def calculate_recall(model_name, df, ground_truth_genres):
    """Calculates and stores recall at positions 1, 5, and 10 for each result
       column. Also, counts how many times each word was replaced using the
       "replacements" dictionary.

    Args:
        model_name (str): Name of the model.
        df (DataFrame): Input DataFrame.
        ground_truth_genres (list of str): Ground truth genres.

    Returns:
        df (DataFrame): DataFrame with recall values for each result column.
        movie_recall (Series): Average recall per movie.
        replacements_counter (dict): Dictionary with counts of each word
```

```
            replaced.
    """
    replacements_counter = {key: 0 for key in REPLACEMENTS.keys()}

    for result_column in RESULT_COLUMNS:  # For each results column.
        for i in range(len(df)):  # For each movie.
            try:  # If the current predictions are not nan.
                predicted_genres = df.at[i, result_column].split("|")

            # If the current predictions are nan, save nan values for the
            # corresponding recall values, skip the rest of the current loop.
            except Exception as e:
                for recall in RECALLS:  # For each recall level.
                    df.at[i, f"R@{recall}_{result_column}"] = nan
                continue

            # Ensures the predictions are lower case and contain no whitespace.
            predicted_genres = [
                s.replace(" ", "").lower() for s in predicted_genres]
            # Counts replacements based on REPLACEMENTS.
            for genre in predicted_genres:
                if genre in REPLACEMENTS.keys():
                    replacements_counter[genre] += 1
            # Makes the replacements based on REPLACEMENTS.
            predicted_genres = [
                REPLACEMENTS.get(item, item) for item in predicted_genres]
            current_movie_genres = df.at[i, "genres"].split("|")
            # Calculates recall@1, recall@5 and recall@10.
            for recall in RECALLS:  # For each recall level.
                # Calculates the appropriate recall value.
                if recall == 1 and predicted_genres:
                    df.at[i, f"R@1_{result_column}"] = int(
                        predicted_genres[0] in current_movie_genres)
                else:
                    df.at[i, f"R@{recall}_{result_column}"] = len(
                        [value for value in current_movie_genres if value in predicted_genres[:recall] and value in ground_truth_genres]) / len(current_movie_genres) if current_movie_genres else nan

    # Gathers data in DataFrame.
    df_export = df[[column for column in df.columns if column == "title" or column.startswith("R@")]]
    df_export.to_csv(f"{PARENT_DIR}/Recall/All/{model_name.split('/')[-1]}.csv", index=False)
    # Calculates average recall per movie.
    movie_recall = df.set_index("title")[
        [column for column in df.columns if column.startswith("R@1_")]].mean(axis=1)

```

```python
    return df, movie_recall, replacements_counter

def calculate_counts(df):
    """Calculates prediction word counts and average recall across all prompts.

    Args:
        df (DataFrame): Recall scores.

    Returns:
        stats (dict): Average recall values.
        prediction_counts (dict): Word prediction counts.
    """
    recall_columns = []
    for column in RESULT_COLUMNS:
        # Covers "0" column.
        if column == "0":
            for recall in RECALLS:  # For each recall level.
                recall_columns.append(f"R@{recall}_0")

        # Covers "1a-24a" and "1b-24b" columns.
        elif column.endswith("a") or column.endswith("b"):
            for recall in RECALLS:  # For each recall level.
                recall_columns.append(f"R@{recall}_{column}")

        # Covers "25-66" columns
        else:
            for recall in RECALLS:  # For each recall level.
                # Removes "a" suffix for "25-66" range.
                recall_columns.append(f"R@{recall}_{column}")

    # Calculates predicted word counts across all styles.
    prediction_counts = {1: Counter(), 5: Counter(), 10: Counter()}
    for style in RESULT_COLUMNS:  # For each prompt style.
        for recall in RECALLS:  # For each recall level.
            for row in df[style]:  # For every movie.
                # Only calculate for this row if there exists predictions.
                if notnull(row):
                    # Makes necessary replacements based on REPLACEMENTS.
                    predictions = [
                        REPLACEMENTS.get(
                            item, item) for item in row.split("|")[:recall]]
                    # Increments every prediction.
                    for prediction in predictions:
                        prediction_counts[recall][prediction] += 1

    # Normalize the prediction counts.
    for recall in RECALLS:  # For each recall level.
        total_counts = sum(prediction_counts[recall].values())
        prediction_counts[recall] = {prediction: count / total_counts for prediction, count in prediction_counts[recall].items()}

```

```
    # Calculates average accuracy per recall.
    avg_recall = {column: df[column].mean() for column in recall_columns}
    avg_recall_combined = {f"average_R@{recall}": sum(v for k, v in avg_recall.items() if f"R@{recall}" in k) / len([k for k in avg_recall.keys() if f"R@{recall}" in k]) for recall in RECALLS}

    # Returns these statistics.
    stats = {**avg_recall, **avg_recall_combined}
    return stats, prediction_counts

def calculate_genre_error_matrix(df, recall_at, ground_truth_genres):
    """Calculates error matrix for each genre without normalization but with
       custom division and rounding.

    Args:
    df (DataFrame):
    recall_at (int): Level of recall to calculate error matrix at
       (1, 5, or 10).
    ground_truth_genres (list of str): Ground truth genres.

    Returns:
    error_matrix (dict of dict): Dictionary of dictionaries representing the
       error matrix.
    """
    # Predefine the error matrix structure with genres in correct order.
    error_matrix = {true_genre: {predicted_genre: 0 for predicted_genre in ground_truth_genres} for true_genre in ground_truth_genres}

    for result_column in RESULT_COLUMNS:  # For each results column.
        for i in range(len(df)):  # For each movie.
            predicted_genres = df.at[i, result_column]
            current_movie_genres = df.at[i, "genres"]
            # Handles nan values.
            if isna(predicted_genres):
                predicted_genres = ""
            if isna(current_movie_genres):
                current_movie_genres = ""

            # Make necessary replacements based on REPLACEMENTS.
            predicted_genres_list = [REPLACEMENTS.get(
                item, item) for item in predicted_genres.split("|")[:recall_at]]
            # Filter predicted words for ground truth genres.
            predicted_genres_list = [
                genre for genre in predicted_genres_list if genre in ground_truth_genres]

            # For each current movie's genre.
            for true_genre in current_movie_genres.split("|"):
                # For each predicted genre.
                for predicted_genre in predicted_genres_list:
```

```python
                    # Increment the appropriate value.
                    error_matrix[true_genre][predicted_genre] += 1

    # Calculates total for each column and adds it to the dictionary.
    for true_genre in ground_truth_genres:
        total_for_genre = sum(error_matrix[true_genre].values())
        error_matrix[true_genre]['total'] = total_for_genre

    # Divides all values by 91 to get average errors for a single prompt across
    # the whole dataset.
    for true_genre in ground_truth_genres:
        for predicted_genre in ground_truth_genres + ['total']:
            error_matrix[true_genre][predicted_genre] = round(
                error_matrix[true_genre][predicted_genre] / 91)

    return error_matrix

def calculate_single_genre_error_matrix(df, result_column, recall_at, ground_truth_genres):
    error_matrix = {true_genre: {predicted_genre: 0 for predicted_genre in ground_truth_genres} for true_genre in ground_truth_genres}

    for i in range(len(df)):
        predicted_genres = df.at[i, result_column]
        current_movie_genres = df.at[i, "genres"]

        # Handles nan values.
        if isna(predicted_genres):
            predicted_genres = ""
        if isna(current_movie_genres):
            current_movie_genres = ""

        # Make necessary replacements based on REPLACEMENTS.
        predicted_genres_list = [REPLACEMENTS.get(item, item) for item in predicted_genres.split("|")[:recall_at]]
        # Filter predicted words for ground truth genres.
        predicted_genres_list = [genre for genre in predicted_genres_list if genre in ground_truth_genres]

        # For each current movie's genre.
        for true_genre in current_movie_genres.split("|"):
            # For each predicted genre.
            for predicted_genre in predicted_genres_list:
                # Increment the appropriate value.
                error_matrix[true_genre][predicted_genre] += 1

    # Calculates total for each column and adds it to the dictionary.
    for true_genre in ground_truth_genres:
        total_for_genre = sum(error_matrix[true_genre].values())
        error_matrix[true_genre]['total'] = total_for_genre

    return error_matrix
```

```python



def main():
    calculate_runtimes()

    all_stats = []
    word_counts = {1: [], 5: [], 10: []}
    rename_dict = {
        "bert-base-uncased": "BERT",
        "roberta-large": "RoBERTa Large",
        "facebook/bart-large": "BART Large",
        "albert-large-v2": "ALBERT Large v2"
    }
    prompt_styles = ["Original", "Custom", "Translated",
                     "Paraphrased", "Thesaurus"]
    movie_recalls = []
    all_replacements = {}

    for llm in LLM_NAMES:  # For each LLM.
        # Merges all prompt styles into one DataFrame.
        for style in prompt_styles:
            filename = f"{PARENT_DIR}/Predictions/{style}/{llm.split('/')
                [-1]}.csv"
            df = read_csv(filename)
            if style == "Original":
                all_predictions = df
            else:
                df.drop(["genres", "title"], axis=1, inplace=True)
                all_predictions = all_predictions.join(df)

        # Renames prompts 25a-46a to 25-46.
        rename_dict = {f"{i}a": str(i) for i in range(25, TOTAL_STYLES)}
        all_predictions.rename(columns=rename_dict, inplace=True)

        # Imports ground truth genres.
        unique_genres_df = read_csv(f"{PARENT_DIR}/Dataset/unique_genres.
            csv")
        ground_truth_genres = unique_genres_df["Unique_Genres"].tolist()
        ground_truth_genres = [s.lower() for s in ground_truth_genres]

        # Calculates recall levels, average recall per movie, and number
            of
        # genre replacements made.
        recall_df, movie_recall, replacements_counter = calculate_recall(
            llm, all_predictions, ground_truth_genres)
        # Stores data for the current model.
        all_replacements[llm] = replacements_counter
        movie_recalls.append(movie_recall)

        # Calculates average recalls and predicted word counts.
        stats, prediction_counts = calculate_counts(recall_df)
        # Adds current LLM's statistics to "all_stats".
```

```
        stats_df = DataFrame(stats, index=[llm])
        all_stats.append(stats_df)

        for recall in RECALLS:  # For each recall level.
            # Calculates predicted word counts.
            genre_counts_df = DataFrame.from_dict(prediction_counts[recall
               ],
                                                   orient="index")
            genre_counts_df.columns = ["Count"]
            genre_counts_df = genre_counts_df.fillna(0)
            genre_counts_df.columns = MultiIndex.from_product(
                [[llm], genre_counts_df.columns])
            word_counts[recall].append(genre_counts_df)

            # Calculates error matrix.
            error_matrix = calculate_genre_error_matrix(
                recall_df, recall, ground_truth_genres)
            # Saves error matrix to CSV file.
            error_matrix_df = DataFrame(error_matrix).fillna(0)
            filename = f"{PARENT_DIR}/Results/Error Matrices/{llm.split
               ('/')[-1]}/average_{recall}.csv"
            error_matrix_df.to_csv(filename)



            for prompt_style in RESULT_COLUMNS:

                # Calculate error matrix for the current prompt style
                   column
                error_matrix = calculate_single_genre_error_matrix(
                   all_predictions, prompt_style, recall,
                   ground_truth_genres)

                # Save the error matrix to a CSV file
                error_matrix_df = DataFrame(error_matrix).fillna(0)

                # Create directory for LLM and recall level if it doesn't
                   exist
                directory = f"{PARENT_DIR}/Results/Error Matrices/{llm.
                   split('/')[-1]}/{recall}"
                if not os.path.exists(directory):
                    os.makedirs(directory)

                filename = f"{directory}/{prompt_style}.csv"
                error_matrix_df.to_csv(filename)



    # Saves genre replacements counter to CSV file.
    all_replacements_df = DataFrame(all_replacements)
    # Divides each value by 91 and then rounds it.
    all_replacements_df = all_replacements_df.divide(91).round(0)
    all_replacements_df.loc["Total"] = all_replacements_df.sum(axis=0)
```

```
    all_replacements_df["Total"] = all_replacements_df.sum(axis=1)
    filename = f"{PARENT_DIR}/Results/Summaries/replacements_counts.csv"
    all_replacements_df.to_csv(filename)

    # Saves all statistics to CSV file.
    all_stats_df = concat(all_stats, axis=0).reset_index()
    all_stats_df.rename(columns={"index":"llm"}, inplace=True)
    filename = f"{PARENT_DIR}/Results/Summaries/recall_stats.csv"
    all_stats_df.to_csv(filename, index=False)

    # Saves movie recalls to CSV file.
    movie_recalls_df = concat(movie_recalls, axis=1)
    movie_recalls_df.columns = LLM_NAMES
    # Add average recall per movie column.
    movie_recalls_df["Average"] = movie_recalls_df.mean(axis=1)
    filename = f"{PARENT_DIR}/Results/Summaries/movie_recalls.csv"
    movie_recalls_df.to_csv(filename)

    # Saves all predicted word counts to CSV file.
    for recall in RECALLS:  # For each recall level.
        word_counts_df = concat(word_counts[recall], axis=1)
        word_counts_df.columns = [col[0] for col in word_counts_df.columns
            ]
        filename = f"{PARENT_DIR}/Results/Prediction Counts/R@{recall}.csv
            "
        word_counts_df.to_csv(filename)

        # Filters predicted word counts to only include ground truth
            genres.
        valid_genres = [genre for genre in ground_truth_genres if genre in
             word_counts_df.index]
        genre_counts_df = word_counts_df.loc[valid_genres]
        # Renormalize the genre counts.
        genre_counts_df = genre_counts_df.divide(genre_counts_df.sum(axis
            =0), axis=1)
        # Save the genre counts to CSV file.
        filename = f"{PARENT_DIR}/Results/Genre Counts/R@{recall}.csv"
        genre_counts_df.to_csv(filename)

if __name__ == "__main__":
    main()
```

B.6 stats_eval_intermediate.py

```python
# stats_eval_intermediate.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Statistically evaluates the intermediate movie prediction results.
#

# Imports.
from pandas import read_csv, concat, DataFrame
from pandas.errors import PerformanceWarning
from pathlib import Path
from warnings import filterwarnings

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"
RESULT_COLUMNS = ["cast_a", "cast_b", "director_a", "director_b", "producer_a",
                  "producer_b", "screenwriter_a", "screenwriter_b",
                  "composer_a", "composer_b", "editor_a", "editor_b",
                  "year_a", "year_b", "distributor_a", "distributor_b",
                  "country_a", "country_b"]

# Disables relevant warnings.
filterwarnings("ignore", category=PerformanceWarning)

def calculate_recall(model_name, df, actual_genres):
    """Calculates and store recall at positions 1, 5, and 10 for each result
       column.

    Args:
        model_name (str): Name of the model.
        df (DataFrame): Input DataFrame.
        ground_truth_genres (list of str): Ground truth genres.

    Returns:
        df (DataFrame): DataFrame with recall values for each result column.
    """
    for result_column in RESULT_COLUMNS:  # For each results column.
        for i in range(len(df)):
            try:
                # Gets genres.
                predicted_genres = df.at[i, result_column].split("|")
                predicted_genres = [
                    s.replace(" ", "").lower() for s in predicted_genres]
                replacements = {"music": "musical", "romantic": "romance",
                                "comedic": "comedy", "comedies": "comedy",
                                "animated": "animation", "love": "romance"}
                predicted_genres = [
```

```python
                    replacements.get(item, item) for item in predicted_genres]
                genre_truths = df.at[i, "genres"].split("|")
                # Calculates recall@1, recall@5 and recall@10.
                df.at[i, f"R@1_{result_column}"] = int(
                    predicted_genres[0] in genre_truths) if predicted_genres else 0
                df.at[i, f"R@5_{result_column}"] = len(
                    [value for value in genre_truths if value in predicted_genres[:5] and value in actual_genres]) / len(genre_truths)
                df.at[i, f"R@10_{result_column}"] = len(
                    [value for value in genre_truths if value in predicted_genres and value in actual_genres]) / len(genre_truths)

            except: print(model_name, result_column, i)

    # Exports DataFrame.
    df_export = df[[column for column in df.columns if column == "title" or column.startswith("R@")]]
    filename = f"{PARENT_DIR}/Recall/Intermediate/{model_name.split('/')[-1]}.csv"
    df_export.to_csv(filename, index=False)

    return df

def calculate_stats(df):
    """Calculates and average recall across all prompts.

    Args:
        df (DataFrame): Recall scores.

    Returns:
        stats (dict): Average recall values.
    """
    recall_columns = [f"R@{i}_{j}" for i in [1,5,10] for j in RESULT_COLUMNS]

    # Calculates average accuracy per recall.
    avg_recall = {column: df[column].mean() for column in recall_columns}
    avg_recall_1 = (sum([v for k, v in avg_recall.items() if "R@1" in k]) /
                   len([k for k in avg_recall.keys() if "R@1" in k]))
    avg_recall_5 = (sum([v for k, v in avg_recall.items() if "R@5" in k]) /
                   len([k for k in avg_recall.keys() if "R@5" in k]))
    avg_recall_10 = (sum([v for k, v in avg_recall.items() if "R@10" in k]) /
                    len([k for k in avg_recall.keys() if "R@10" in k]))

    # Returns these statistics.
```

```python
    stats = {**avg_recall, "average_R@1": avg_recall_1,
             "average_R@5": avg_recall_5,
             "average_R@10": avg_recall_10}
    return stats

def main():
    all_stats = []
    llm_dict = {
        "bert-base-uncased": "BERT",
        "roberta-large": "RoBERTa Large",
        "facebook/bart-large": "BART Large",
        "albert-large-v2": "ALBERT Large v2"
    }

    for llm in list(llm_dict.keys()):  # For each LLM.

        # Defines intermediate prompt styles df, if needed.
        filename = f"{PARENT_DIR}/Predictions/Intermediate/{llm.split('/')[-1]}.csv"
        all_predictions_df = read_csv(filename)

        # Imports ground truth genres.
        unique_genres_df = read_csv(f"{PARENT_DIR}/Dataset/unique_genres.csv")
        actual_genres = unique_genres_df["Unique_Genres"].tolist()
        actual_genres = [s.lower() for s in actual_genres]

        # Calculates recall levels.
        recall_df = calculate_recall(llm, all_predictions_df, actual_genres)

        stats = calculate_stats(recall_df)
        # Renames the LLMs.
        llm = llm_dict.get(llm, llm)
        # Adds current LLM's statistics to "all_stats".
        stats_df = DataFrame(stats, index=[llm])
        all_stats.append(stats_df)

    # Saves all statistics to CSV file.
    all_stats_df = concat(all_stats, axis=0).reset_index()
    all_stats_df.rename(columns={"index":"llm"}, inplace=True)
    all_stats_df.to_csv(f"{PARENT_DIR}/Results/Intermediate/recall_stats.csv",
                        index=False)

if __name__ == "__main__":
    main()
```

B.7 t_tests.py

```python
# t_tests.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Runs statistical significance tests for different recalls and prompt
    styles
# for each movie.
#

# Imports.
from pandas import read_csv, concat, Series, DataFrame
from scipy import stats
from pathlib import Path

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"

def t_tests(base_prompt):
    """Performs paired t-tests between 2 sets of predictions.

    Args:
        base_prompt (str): Column name of what column to use as the base
        column for the significance tests.
    """
    results = DataFrame(columns=["LLM", "Recall", "Prompt", "Mean
       Difference",
                                 "Test Statistic", "P-Value"])
    best_prompt_results = DataFrame(columns=["LLM", "Recall", "Best Prompt
       ",
                                             "Max Mean Difference",
                                             "Test Statistic", "P-Value"])

    for llm in ["bert-base-uncased", "roberta-large",
                "bart-large", "albert-large-v2"]:  # For each LLM.
        # Reads recall results.
        df = read_csv(f"{PARENT_DIR}/Recall/All/{llm}.csv")

        # Iterates over different recall levels.
        for recall in ["R@1", "R@5", "R@10"]:
            max_mean_diff = float("-inf")
            best_prompt, best_t_stat, best_p_val = None, None, None
            # Selects base prompt column to perform t-test with.
            base_column = f"{recall}_{base_prompt}"

            for prompt in range(1, 67):  # Iterates over enriched prompts.
                for style in ["a", "b"]:
                    current_df = df.copy()

                    # Skips "b" styles where they don't exist.
                    if prompt in ([0] + list(range(24, 67))) and style ==
                       "b":
```

```
                        continue
                    if prompt > 24: style = ""
                    # Selects current best-performing prompt column.
                    best_performing_column = f"{recall}_{prompt}{style}"
                    # Drops rows where the best-performing prompt has null
                    # values.
                    current_df.dropna(subset=[best_performing_column],
                                      inplace=True)
                    # Performs paired t-test.
                    t_stat, p_val = stats.ttest_rel(
                        current_df[base_column],
                        current_df[best_performing_column],
                        alternative="less")
                    # Calculates mean difference between the 2 sets of data.
                    mean_diff = (current_df[best_performing_column].mean() -
                                 current_df[base_column].mean())

                    # Stores the result.
                    result = Series([llm, recall, f"{prompt}{style}",
                                     mean_diff, t_stat, p_val])
                    results = concat([results, result], axis=1)

                    # Updates the best-performing prompt style variables if the
                    # current mean difference is greater.
                    if mean_diff > max_mean_diff:
                        max_mean_diff = mean_diff
                        best_prompt = f"{prompt}{style}"
                        best_t_stat = t_stat
                        best_p_val = p_val

            # Stores the best prompt style result.
            best_prompt_result = Series([llm, recall, best_prompt,
                                         max_mean_diff, best_t_stat,
                                         best_p_val])
            best_prompt_results = concat([best_prompt_results,
                                          best_prompt_result], axis=1)

    # Transposes and cleans up the full results DataFrame.
    results = results.T
    results.columns = ["LLM", "Recall", "Prompt", "Mean Difference",
                       "Test Statistic", "P-Value"]
    results = results[results["LLM"].notna()]
    results.reset_index(drop=True, inplace=True)
    # Saves the results in a CSV file.
    results.to_csv(f"{PARENT_DIR}/Results/T-Tests/{base_prompt}.csv",
                   index=False)

    # Transposes and clean up the best-performing prompt results DataFrame.
    best_prompt_results = best_prompt_results.T
```

```
    best_prompt_results.columns = ["LLM", "Recall", "Best Prompt",
                                   "Max Mean Difference",
                                   "Test Statistic", "P-Value"]
    best_prompt_results = best_prompt_results[
        best_prompt_results["LLM"].notna()]
    best_prompt_results.reset_index(drop=True, inplace=True)
    # Saves the best-performing prompt results in a CSV file.
    filename = f"{PARENT_DIR}/Results/T-Tests/best_performing_prompts_{
       base_prompt}.csv"
    best_prompt_results.to_csv(filename, index=False)

def main():
    t_tests("0")
    t_tests("2a")
    t_tests("2b")
    t_tests("9b")

if __name__ == "__main__":
    main()
```

B.8 graphs.py

```
# graphs.py
# Daniel Van Cuylenburg (k19012373)
# 15/08/2023
#
# Generates graphs used in the report.
#

# Imports.
from pandas import read_csv
from seaborn import heatmap, set
from os import listdir
from numpy import arange
from pathlib import Path
import matplotlib.pyplot as plt

# Constants.
PARENT_DIR = Path(__file__).parent.parent / "Data"

def process_data(data):
    """Separates appropriate recall columns.

    Args:
        data (DataFrame): All predictions data.

    Returns:
        DataFrame: Appropriate recall columns.
    """
    data.set_index("llm", inplace=True)

    recall_1_cols = [col for col in data.columns if "R@1_" in col and "
        R@10" not in col]
    recall_5_cols = [col for col in data.columns if "R@5_" in col]
    recall_10_cols = [col for col in data.columns if "R@10_" in col]

    recall_1 = data[recall_1_cols]
    recall_5 = data[recall_5_cols]
    recall_10 = data[recall_10_cols]

    # Removes the "R@_" part from the column names.
    recall_1.columns = recall_1.columns.str.replace("R@1_", "")
    recall_5.columns = recall_5.columns.str.replace("R@5_", "")
    recall_10.columns = recall_10.columns.str.replace("R@10_", "")

    return recall_1, recall_5, recall_10

def generate_heatmap(data, option):
    """Generates heatmaps.

    Args:
        data (DataFrame): Data to display.
        option (int): Prompts to display:
```

```python
            1 = prompts 0-24,
            2 = prompts 25-66
    """
    # Sets figure size.
    set(rc={"figure.figsize":(14.0, 10.0)})
    # Orders columns appropriately.
    cols_order = ["0"]
    cols_order += [f"{i}{suffix}" for i in range(1, 25) for suffix in ["a", "b"]]
    cols_order += [str(i) for i in range(25, 67)]
    data = data[cols_order]

    # Selects columns to display.
    if option == 1:
        columns_to_drop = [f"{i}" for i in range(25, 67)]
    else:
        columns_to_drop = ["0"] + [f"{i}a" for i in range(1, 25)] + [f"{i}b" for i in range(1, 25)]
    data = data.drop(columns=columns_to_drop)

    # Generates heatmap.
    ax = heatmap(data, cmap="YlGnBu", annot=False, cbar=False)  # Save heatmap object in ax
    plt.ylabel("Large Language Model")

    # Sets threshold used to determine text colour for readability.
    threshold = data.max().max()/2
    fontsize = 11
    # Adds annotations.
    for i in range(data.shape[0]):  # For each row.
        for j in range(data.shape[1]):  # For each column.
            # If the value is the highest in the row, set text colour to red.
            if round(data.iloc[i, j], 3) == round(data.iloc[i].max(), 3):
                plt.text(j+0.5, i+0.5, f"{data.iloc[i, j]:.3f}",
                         horizontalalignment="center",
                         verticalalignment="center",
                         fontweight="bold",
                         color="red",
                         fontsize=fontsize,
                         rotation=90)
            # Else, text colour should be black or white.
            else:
                text_color = "black" if data.iloc[i, j] < threshold else "white"
                plt.text(j+0.5, i+0.5, f"{data.iloc[i, j]:.3f}",
                         horizontalalignment="center",
                         verticalalignment="center",
                         color=text_color,
                         fontsize=fontsize,
                         rotation=90)

    # Adds appropriate separator lines.
```

```
    if option == 1:
        label = plt.xlabel("" + "\n" + " "*50 +
           "Custom Prompts" + " "*140 + "Original Prompts"+ " "*0)
        col_index = list(data.columns).index("9b")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
    else:
        label = plt.xlabel("" + "\n" +
            "Thesaurus-Paraphrased Prompts" + " "*50 + "T5-Paraphrased Prompts" + " "*30 + "Translated Prompts" + " "*55 + "Naturally Worded Prompts")
        col_index = list(data.columns).index("36")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("46")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("51")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("54")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("57")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("60")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
        col_index = list(data.columns).index("63")
        ax.vlines(col_index+1, *ax.get_ylim(), colors="black", linestyles="dashed", linewidth=2)
    label.set_rotation(180)

    # Rotates x-axis ticks.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

    # Shows figure.
    plt.show()
    plt.close()
    plt.clf()

def generate_heatmap_intermediate(data, title):
    """Generates intermediate heatmaps.

    Args:
        data (DataFrame): Data to display.
        option (int): Prompts to display:
            1 = prompts 0-24,
            2 = prompts 25-66
    """
    # Sets figure size.
    set(rc={'figure.figsize':(11.0, 8.0)})
```

```python

    # Calculate column-wise average and append it to the DataFrame.
    data.loc['Average'] = data.mean()

    # Generates heatmap.
    ax = heatmap(data, cmap="YlGnBu", annot=False, cbar=False)  # Save heatmap object in ax

    plt.ylabel('Large Language Model')

    # Sets threshold used to determine text colour for readability.
    threshold = data.max().max()/2
    # Adds annotations.
    for i in range(data.shape[0]):  # For each row.
        for j in range(data.shape[1]):  # For each column.
            # If the value is the highest in the row, set text colour to red.
            if round(data.iloc[i, j], 2) == round(data.iloc[i].max(), 2):
                plt.text(j+0.5, i+0.5, f'{data.iloc[i, j]:.2f}',
                         horizontalalignment='center',
                         verticalalignment='center',
                         fontweight='bold',
                         color='red',
                         fontsize=10)
            # Else, text colour should be black or white.
            else:
                text_color = 'black' if data.iloc[i, j] < threshold else 'white'
                plt.text(j+0.5, i+0.5, f'{data.iloc[i, j]:.2f}',
                         horizontalalignment='center',
                         verticalalignment='center',
                         color=text_color,
                         fontsize=10)

    # Rotates x-axis ticks.
    ax.set_xticklabels(ax.get_xticklabels(), rotation=-45)

    # Shows figure.
    plt.show()
    plt.close()
    plt.clf()

def mean_diff_bar_chart(data, title):
    """Plots mean difference bar charts.

    Args:
        data (Dict): Data to plot bar chart with.
        title (str): Title to use for file.
    """
    colors = ["b", "g", "r", "c", "m", "y", "k"]
    # Creates a list of models.
    models = list(data.keys())
    # Creates a dictionary mapping each model to a color.
```

```python
    color_dict = {model: color for model, color in zip(models, colors)}
    # Sorts the dictionary in descending order of the values.
    data = {k: v for k, v in sorted(data.items(), key=lambda item: item[1], reverse=True)}
    # Creates an array with the positions of each bar on the x-axis.
    x_pos = arange(len(data))
    # Increases the size of the plot (width=10, height=6).
    plt.figure(figsize=(10, 6))
    # Creates the bar chart assigning the same color to each model every time.
    plt.bar(x_pos, list(data.values()), color=[color_dict[key] for key in data.keys()])
    # Changes the bar labels on x-axis and rotate labels by 45 degrees.
    plt.xticks(x_pos, list(data.keys()), rotation=-20)
    # X and y-axis labels.
    plt.xlabel("Large Language Models")
    plt.ylabel("Mean Difference")
    # Shows the figure.
    plt.tight_layout()
    plt.savefig(f"{PARENT_DIR}/Graphs/{title}_all.png")

def csv_to_latex_table(csv_filename, output_filename):
    """Converts CSV files to latex tables for the report.

    Args:
        csv_filename (str): Name of CSV file to convert.
        output_filename (str): Name of output filename.
    """
    # Load CSV into Pandas DataFrame
    df = read_csv(csv_filename, index_col=0)

    # Create LaTeX table
    with open(output_filename, "w") as f:
        for idx, row in df.iterrows():
            latex_row = "{} & ".format(idx) + " & ".join(map(lambda x: "{:.0f}".format(float(x)), row))
            latex_row += " \\\\\hline\n"
            f.write(latex_row)

def main():
    data = read_csv(f"{PARENT_DIR}/Results/Summaries/recall_stats.csv")
    data_intermediate = read_csv(f"{PARENT_DIR}/Results/Intermediate/recall_stats.csv")

    recall_1, recall_5, recall_10 = process_data(data)
    generate_heatmap(recall_1, 1)
    generate_heatmap(recall_5, 1)
    generate_heatmap(recall_10, 1)

    generate_heatmap(recall_1, 2)
    generate_heatmap(recall_5, 2)
    generate_heatmap(recall_10, 2)

```

```
    recall_1, recall_5, recall_10 = process_data(data_intermediate)
    generate_heatmap_intermediate(recall_1, 'R@1')
    generate_heatmap_intermediate(recall_5, 'R@5')
    generate_heatmap_intermediate(recall_10, 'R@10')

    # For each error matrix, convert it into a Latex table.
    for llm in ["bert-base-uncased", "roberta-large",
                "bart-large", "albert-large-v2"]:
        for filename in listdir(f"{PARENT_DIR}/Results/Error Matrices/{llm
            }"):
            if filename.endswith(".csv"):
                filepath = f"{PARENT_DIR}/Results/Error Matrices/{llm}/{
                    filename}"
                csv_to_latex_table(filepath, f"{PARENT_DIR}/Graphs/Latex/
                    Error Matrices/{llm}_{filename[:-4]}.txt")

    # For each genre count table, convert it into a Latex table.
    for filename in listdir(f"{PARENT_DIR}/Results/Genre Counts/"):
        if filename.endswith(".csv"):
            filepath = f"{PARENT_DIR}/Results/Genre Counts/{filename}"
            csv_to_latex_table(filepath, f"{PARENT_DIR}/Graphs/Latex/Genre
                Counts/{filename[:-4]}.txt")

    # R@1
    data = {
        "BERT (50)": 0.365,
        "RoBERTa large (50)": 0.334,
        "BART large (50)": 0.458,
        "ALBERT large v2 (50)": 0.428
    }
    mean_diff_bar_chart(data, "mdiff_1")

    # R@5
    data = {
        "BERT (50)": 0.304,
        "RoBERTa large (9b)": 0.373,
        "BART large (50)": 0.586,
        "ALBERT large v2 (50)": 0.604
    }
    mean_diff_bar_chart(data, "mdiff_5")

    # R@10
    data = {
        "BERT (50)": 0.296,
        "RoBERTa large (9b)": 0.303,
        "BART large (50)": 0.683,
        "ALBERT large v2 (50)": 0.735
    }
    mean_diff_bar_chart(data, "mdiff_10")


if __name__ == "__main__":
    main()
```