Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Digital Engineering

Investigating Transformer Input Embedding in Neural Translation Language Model for Search

Master's Thesis

Eyasu Berhane Habte

1. Referee: Prof. Dr. Benno Stein
2. Referee: Jun. Prof. Dr. Harry Scells

Submission date: April 10, 2025

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, Germany, April 10, 2025

. . . . . .
Eyasu Berhane Habte

Abstract

Information retrieval (IR) is the process of searching for relevant information from large collections based on users' needs. One of the challenges in IR is the vocabulary gap, where discrepancies between query and document terms lower retrieval effectiveness. This thesis investigates the Neural Translation Language Model (NTLM) approach to address this challenge, leveraging an input embedding from the transformer models. Originally, the NTLM approach was implemented using Word2Vec embeddings. This thesis extends the NTLM by incorporating transformer input embeddings, aiming to improve reranking effectiveness. We compare our NTLM approach to the Dirichlet-smoothed language model for first-stage retrieval and evaluate its effectiveness across diverse datasets. Additionally, we assess the NTLM effectiveness against the WordLlama method [23], a lightweight NLP toolkit for effective document reranking through semantic cosine similarity.

We conduct multiple experiments using the PyTerrier platform for indexing, initial ranking, and evaluation. We use ten benchmark datasets from the BEIR collection [16], a benchmark suite for evaluating retrieval models on a diverse range of datasets and the `ir_datasets` [1] [20] that provides standardized access to many IR datasets. The effectiveness is assessed using standard IR evaluation measures. The results indicate that the Dirichlet model, with a well-tuned smoothing parameter μ, maintains high effectiveness, while reranking with the NTLM often leads to lower effectiveness in most cases. Among the NTLM models, MonoBERT input embedding was effective, while others like Word2Vec, GPT, and RoBERTa were less so. In contrast, the WordLlama method shows better overall effectiveness, making it a strong candidate for document reranking tasks.

A key reason for the NTLM lower effectiveness lies in its word-to-word translation quality, which directly affects its reranking effectiveness. Analyzing word translations reveals that while most embeddings accurately translate common words, differences emerge with less frequently occurring terms. The Skip-Gram (Google News) model is consistently more effective than others in handling less frequent terms. The MonoBERT and RoBERTa input embedding use subword tokenization to process unknown words effectively, whereas GPT-2 input embedding, despite its large vocabulary, shows the lowest effectiveness. These findings highlight the impact of embedding architectures and training data on word translation tasks, directly influencing the NTLM effectiveness in document ranking.

This thesis demonstrates how the input embedding can be integrated into the NTLM approach, providing insights into the relationship between embedding characteristics and document ranking effectiveness. While the Dirichlet smoothed model remains a strong initial ranker, often more effective than the NTLM approach using different word embedding and input embedding across datasets. The word embeddings still require further refinement to improve semantic understanding and document ranking effectiveness using the NTLM method. Future work could investigate integrating contextualized embeddings and neural query expansion techniques into the NTLM, potentially enhancing its ability to capture query-document interactions and improve ranking effectiveness.

[1] https://ir-datasets.com/

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information retrieval (IR) systems play an important role in our increasingly digital world, giving access to large amounts of information. The design of efficient and effective retrieval models has been a central focus in the field of IR for decades. Over the years, various models have been proposed and implemented, each aiming to improve the effectiveness and efficiency of IR. In the early development of IR, vector space models [33, 34] and probabilistic models [12, 28] emerged as dominant approaches, primarily due to their ability to represent and compare documents and queries in a way to compute and express them mathematically. These foundational methods established the groundwork for subsequent innovations in the field.

A significant advancement in IR performance came with the introduction of language modeling techniques [28, 43], which proposed a probabilistic approach that ranks documents based on the likelihood of a query being generated from each document's model. These document models are estimated using multinomial distributions, with smoothing techniques ensuring accurate estimation. Despite their empirical success, language models face the ongoing challenge of addressing the vocabulary gap, where a mismatch between the terms used in queries and documents can impact retrieval effectiveness. The vocabulary gap arises from the natural disparity between the user queries and the extensive, diverse language used in relevant documents. This mismatch can impact retrieval effectiveness, as relevant documents may be overlooked due to terminological differences. To address this issue, statistical translation language models [2] were proposed drawing inspiration from the field of statistical machine translation [6].

Translation language models aim to estimate the likelihood of translating a document into a query, assigning probabilities to term translations. This approach directly mitigates the vocabulary gap problem by allowing for semantic connections between different terms. The process of query formulation can be viewed as a form of translation: from the vast language of documents to the concise language of queries. This perspective frames the IR problem as a translation task, where the goal is to bridge the gap between document and query languages. However, a primary challenge in applying translation language models to IR is the accurate estimation of the true probability that a query could be generated as a translation of a document.

Translation language model approaches to this problem have relied on statistical and probabilistic methods. However, recent advancements in machine learning, particularly in neural networks, have opened new methods for addressing this challenge. The emergence of word embedding techniques, like Word2Vec [22], has advanced how we represent and process textual data. Word embeddings capture semantic relationships between words in a dense vector space, allowing for more detailed comparisons between terms. Specifically, two such models, the continuous bag-of-words model and the skipgram model, produce vector representations of words that have shown effectiveness on a number of linguistic tasks, including word similarity and word analogy [22].

Building on the success of word embeddings, transformer models [39] have significantly advanced the state of the art in natural language processing. The input embeddings from transformer models offer rich representations of words, capturing semantic similarities and fine-grained details of word usage. The input embedding layer provides these initial word representations before any contextual processing occurs. These embeddings form the initial stage of the transformer architecture, mapping each token to a dense vector in a high-dimensional space. The input embedding layer consists of learned parameters that capture the lexical and semantic properties of words, similar to traditional word embeddings, but with the added benefit of being trained within a larger neural network. Since these embeddings are generated before the self-attention layers, they remain consistent across different contexts, preserving the core meaning of tokens. Moreover, the model's ability to handle subword units enhances the robustness of these embeddings, especially for out-of-vocabulary words and morphological variations.

As different transformer models continue to evolve, the WordLlama model [23] emerges as a promising approach. WordLlama, a lightweight model, extracts token representations from large language models like LLaMA, offering compact, high-quality word embeddings. This makes it a better alternative to traditional models like Word2Vec. By incorporating WordLlama embeddings into the NTLM, we also aim to improve reranking effectiveness.

Extending Zuccon's [46] prior work, which integrates static Word2Vec embeddings into the NTLM approach, we enhance this approach by incorporating transformer input embeddings into the NTLM. By leveraging these embeddings within the translation language model, we aim to address the vocabulary gap and enhance semantic understanding between documents and queries, improving reranking effectiveness. This thesis primarily investigates whether transformer input embeddings achieve higher effectiveness than static word embeddings within the NTLM, while also evaluating the effectiveness of the Dirichlet-smoothed language model as an initial ranker [43]. Although the Dirichlet model has shown its effectiveness as a first-stage ranker, its reliance on exact term matching limits its ability to capture deeper semantic relationships. In contrast, the NTLM enriches translation modeling by incorporating word embedding, providing a more advanced approach to capturing term relationships.

This thesis demonstrates how Word2Vec and transformer input embeddings can be effectively integrated into the NTLM and compares their effectiveness across different datasets. The analysis extends beyond simple effectiveness comparisons, examining how different embedding types and configurations impact the effectiveness of the model on word-to-word translation analysis. The primary focus is on assessing the impact of transformer-

based input embeddings on reranking effectiveness. The research aims to address the following questions:

RQ1: Comparative Analysis of Neural Translation Language Model To what extent do the NTLM using Word2Vec embeddings and the transformer input embeddings improve effectiveness compared to the initial ranker, Dirichlet-smoothed language model, across different datasets? This question investigates whether incorporating word embedding into the translation language model can better capture semantic relationships between queries and documents, potentially offering improved solutions to the vocabulary mismatch problem and improving the reranking of the documents.

RQ2: Impact of Embedding Variations and Fine-Tuned Transformers How do different configurations of Word2Vec models, input embeddings from various transformer architectures, and fine-tuned transformer models like MonoBERT affect retrieval effectiveness? This investigation examines the relationship between embedding types, characteristics, and ranking effectiveness, analyzing how architectural choices, training configurations, and fine-tuning influence the model's ability to capture semantic similarities and improve retrieval and ranking effectiveness.

RQ3: Effectiveness of WordLlama for Reranking and NTLM Enhancement How effectively does the WordLlama model, with its lightweight and efficient embedding derived from large language models (LLMs), enhance the effectiveness of the NTLM and reranking tasks? This question investigates whether WordLlama's compact yet high-quality embeddings, extracted from LLMs like LLaMA 3, can improve semantic representation and reranking effectiveness compared to Word2Vec and transformer input embeddings.

Chapter 2

Background

This chapter provides an overview of key concepts and developments in IR relevant to the thesis. It focuses on the evolution from traditional IR systems to modern neural models, emphasizing their role in IR tasks.

IR has evolved from early well-known models like Salton's vector space model [33] and Robertson's probabilistic model [30], to more advanced statistical approaches. Techniques from speech recognition, such as Shannon's noisy channel and n-gram models [8], laid the foundation for the Translation Language Model (TLM), which estimates query-document relevance through translation probabilities.

In the following decades, IR research has continued to evolve, incorporating more advanced and effective models. For instance, the BM25 model, an extension of the probabilistic approach, has become one of the most widely used ranking models in modern IR systems [29]. The emergence of distributed representations of words, such as Word2Vec [22], marked a significant shift towards leveraging word embeddings to capture semantic relationships, improving the effectiveness of IR models. More recently, Transformer-based models like BERT [11] have improved the field, providing rich representations that support more effective retrieval.

This chapter focuses more explaining the technical foundations necessary for understanding the NTLM. We begin by examining core IR components and retrieval functions that form the basis of modern search systems. The discussion then progresses through the fundamentals of IR models, from traditional approaches to advanced language modeling techniques, with particular attention to the Dirichlet language model and translation language models. A large portion of this chapter focuses on neural approaches in IR, specifically the evolution of vector representations for text. We present dense vector representation methods, including the Skip-gram and Continuous Bag-of-Words (CBOW) architectures, and analyze the input embedding mechanisms of transformer models to improve retrieval and ranking effectiveness.

The chapter concludes with an overview of standard IR evaluation metrics, including Mean Average Precision (MAP), Precision at 10 (P@10), and normalized Discounted Cumulative Gain (nDCG), which provide the quantitative framework for assessing retrieval effectiveness.

2.1 IR Components and Retrieval Functions

The effectiveness of an IR system depends on several key components, including document representation, query representation, and the retrieval function that assesses the relevance of documents to the query [1]. This section explores these components, clarifying their roles and how they contribute to the overall performance of IR systems.

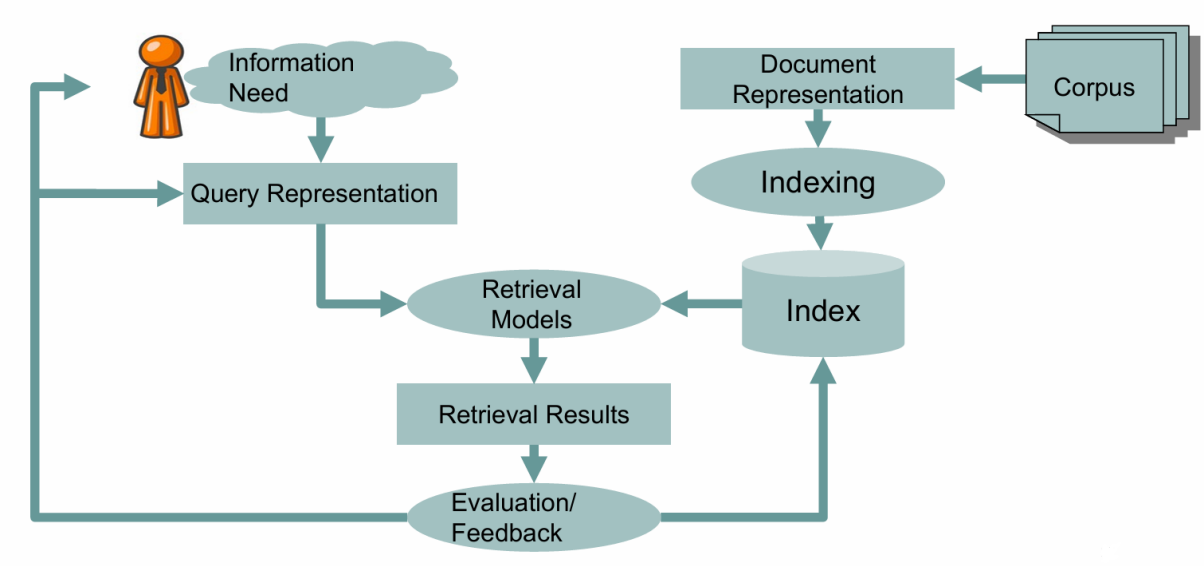

Figure 2.1: IR System.

As shown in the figure the key components of an IR system are:

- Document Representation
- Query Representation
- Retrieval Function

Document Representation: Document representation is a foundational component of IR systems. It involves transforming raw text documents into structured formats that capture their essential features, enabling efficient processing and analysis. The first step in this process is text processing, which includes tokenization, normalization (e.g., lowercasing, stemming), and stopword removal. These steps ensure that the text is clean, consistent, and ready for further analysis. As shown in the conceptual model diagram Figure 2.2, the Document Representation node results from transforming raw documents through processes like indexing and embedding. These processes extract and encode key features, such as term frequencies and semantic relationships, which are critical for matching documents to user queries.

Effective document representation relies on indexing, which creates a structured index of terms or features extracted from documents. A common method is the inverted index, which maps terms to the documents containing them. For example, if a document contains the word “information,” the inverted index stores entries for each term, pointing

back to the document. This enables fast lookup during retrieval, making it a very important component of traditional IR systems.

In modern IR systems, embedding-based representations are widely used. Unlike traditional sparse representations (e.g., term-frequency vectors), embeddings represent documents in a dense vector space, capturing semantic relationships between words and phrases. Techniques like Word2Vec, GloVe, and BERT generate dense vectors that encode semantic information. For instance, the embeddings for "happy," "joyful," and "cheerful" might reflect their close semantic similarity. These embeddings enable the system to understand the underlying meaning of documents, improving retrieval effectivenss and addressing challenges like vocabulary mismatch. Refer the key difference between the indexing and embedding in Table 2.1.

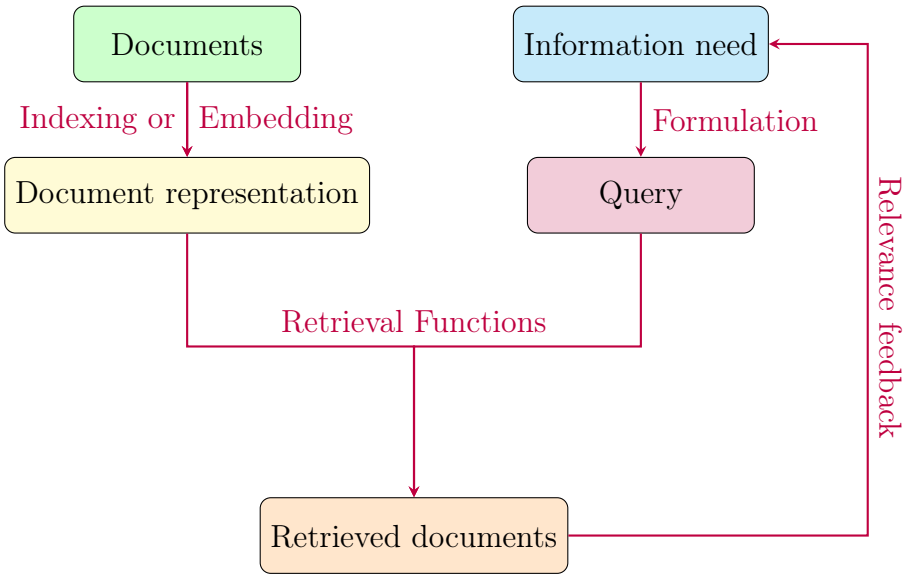

Figure 2.2: Conceptual model for IR

Table 2.1: Comparison of Indexing and Embedding for Document Representation

Aspect	Indexing	Embedding
Representation	Sparse (e.g., term-frequency vectors)	Dense (e.g., neural embeddings)
Semantic Capture	Limited to term-based matching	Captures semantic relationships and context
Scalability	Highly scalable for large datasets	Computationally intensive but more effective
Use Case	Keyword-based searches	Semantic searches

Query Representation: Query representation is an important component of an IR system, as it captures the user's search intent and ensures effective matching of queries to relevant documents. While document representation focuses on structuring raw documents for efficient retrieval, query representation centers on understanding and encoding the user's information needs. This process mirrors document representation but must also account for the variable nature of queries, which can range from simple keywords to complex natural language questions.

Queries, like documents, must be represented in a format that the IR system can process. The key challenge lies in aligning the query representation with the document representation to facilitate effective relevance assessment. This alignment ensures that the retrieval function can process both the query and documents consistently, improving the accuracy of search results. For example, if documents are represented using embeddings (e.g., Word2Vec or BERT), the query must also be encoded in the same embedding space to enable meaningful comparison. This is particularly important for the NTLM, where semantic alignment between queries and documents is essential for addressing challenges like vocabulary mismatch.

Retrieval Function: The retrieval function is the core algorithmic component of an IR system. It evaluates the relevance of documents to a given query and ranks them accordingly, ensuring that the most relevant documents are returned to the user. As shown in the conceptual model diagram Figure 2.2, the Retrieval Functions node connects the Document Representation and Query Representation nodes to the Retrieved Documents node. This shows how the retrieval function acts as the intermediary that bridges queries and documents by performing tasks such as representation alignment, relevance scoring, and ranking. The retrieval function operates through the following key aspects: relevance scoring, efficiency, and scalability. These aspects ensure that the system can effectively match queries to documents while handling large-scale datasets and delivering results with low latency.

Relevance Scoring is the primary task of a retrieval function. It involves computing a relevance score for each document with respect to a query, which quantifies how well the document matches the query. This score allows the system to rank documents in order of relevance, ensuring that the most relevant results are presented to the user at the top of the list. For example, in a search for "climate change impacts", documents discussing the effects of climate change on ecosystems would receive higher relevance scores than those with only indirect mentions of the topic.

Efficiency and Scalability are important for retrieval functions, especially in real-time search systems. They must be computationally efficient to handle large-scale datasets and deliver results with low latency. Techniques like inverted indexing for term-based retrieval and approximate nearest neighbor search for neural retrieval are implemented to ensure that the system can process queries quickly, even when dealing with millions of documents. This efficiency is essential for providing a smooth user experience in modern IR systems.

Different retrieval models use various types of retrieval functions. The IR models implemented in this thesis are based on the following retrieval function:
Probabilistic Retrieval Models: These models rank documents based on the probability that a document is relevant to the query. For example, BIM and BM25 models compute relevance based on term occurrence probabilities [32], as shown in the equation below:

$$\rho(q, d) = P(R = 1|q, d) \tag{2.1}$$

where $\rho(q, d)$ represents the probability that document d is relevant to query q.

Language Models: These models build a probabilistic model of the language in a document. The retrieval function measures the likelihood of generating the query from the document's language model:

$$\rho(q, d) = P(q|d) \tag{2.2}$$

where $\rho(q, d)$ represents the probability of generating the query q from the document language model. Smoothing techniques, such as Dirichlet smoothing, are often used to handle cases where certain terms in the query do not appear in the document.

Neural Retrieval Models: These models differ from traditional approaches by using dense embeddings to represent queries and documents [14, 25].
Word2Vec: It is a shallow neural model that represents words as dense vectors in a continuous vector space [22]. These embeddings capture semantic relationships between words but are static and do not account for context. For example, the word "bank" will have the same embedding regardless of whether it appears in the context of "river bank" or "financial bank." The retrieval function typically uses cosine similarity to measure the similarity between query and document vectors.

Transformer Input Embeddings: These models use the learned token embedding layer from pre-trained transformers like BERT as static word representations [11]. Unlike the contextual representations from the full transformer, this approach uses only the initial embedding table containing fixed vectors for each token in the vocabulary. These embeddings are learned during pre-training and capture semantic relationships similar to Word2Vec, but benefit from the transformer's extensive training data and objectives. For retrieval documents, and queries are represented by aggregating their token embeddings (e.g., by averaging), and similarity is computed using cosine similarity:

$$\text{Cosine Similarity} = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \times ||\vec{d}||}, \tag{2.3}$$

where $\vec{q}$ and $\vec{d}$ are the vector representations of the query and document, respectively.

This section explains the fundamental principles and advancements in IR models. IR has evolved from early traditional methods to current advanced neural approaches. Here, we outline the theoretical foundations and significant innovations in word embedding models, setting the framework for this thesis. Traditional IR models are also discussed, as they

provide essential insights and act as benchmarks for the NTLM approaches. Although many IR models exist, this thesis will focus on those most relevant to our goals.

2.2 Traditional IR models

This section introduces relevant traditional IR models and approaches that serve as foundational baseline for the NTLM. These models provide the theoretical groundwork for formulating the Dirichlet language model and the NTLM.

2.2.1 Language Modelling

The language modeling-based approach [28, 43] to IR ranks documents based on the probability that a document d is relevant to a query q. This probability is expressed as $p(d|q)$. The key idea is to estimate the likelihood of generating the query q from the document language model.

The probability of a document d given a query q can be expressed as:

$$p(d|q) = \frac{p(q|d) \cdot p(d)}{\sum_{\bar{d} \in D} p(q|\bar{d}) \cdot p(\bar{d})} \tag{2.4}$$

$$p(d|q) \propto p(q|d) \cdot p(d) \tag{2.5}$$

$$p(d|q) = p(q|d), \quad \text{assuming } p(d) \text{ is uniform} \tag{2.6}$$

Here, $p(q|d)$ is the likelihood of generating the query q from the document d's language model, and $p(d)$ is the prior probability of the document d. The denominator $\sum_{\bar{d} \in D} p(q|\bar{d}) \cdot p(\bar{d})$ is a normalization factor that ensures the probabilities sum to 1. Under the assumption that $p(d)$ is uniform (i.e., all documents are equally likely), the equation simplifies to $p(d|q) \propto p(q|d)$. This means that the ranking of documents depends solely on the likelihood $p(q|d)$.

The query likelihood $p(q|d)$ is computed as the product of the probabilities of each term t_q in the query q given the document d:

$$p(q|d) = \prod_{t_q \in q} p(t_q|d) \tag{2.7}$$

Here, $p(t_q|d)$ is the probability of the term t_q in the document d, which can be estimated using the maximum likelihood estimate (MLE):

$$p(t_q|d) = \frac{\text{tf}(t_q, d)}{|d|} \tag{2.8}$$

where $\text{tf}(t_q, d)$ is the term frequency of t_q in document d, and $|d|$ is the length of the document (i.e., the total number of terms in d).

However, a major challenge in language modelling is handling terms that appear in the query but not in the document. Without smoothing, such terms would have a probability of zero, leading to a zero likelihood for the entire query. So most approaches to language modeling-based retrieval generally incorporate some form of smoothing [43], which involves sampling terms from both the specific document d and the entire collection D.
The two most commonly used smoothing techniques are:

1. Jelinek-Mercer Smoothing: This technique linearly interpolates between the document language model and the collection language model:

$$p(t_q|d) = \lambda \frac{\text{tf}(t_q, d)}{|d|} + (1 - \lambda) \frac{\sum_{\bar{d} \in D} \text{tf}(t_q, \bar{d})}{\sum_{\bar{d} \in D} |\bar{d}|} \tag{2.9}$$

Where: λ is the interpolation parameter (typically between 0 and 1), which controls the weight given to the document language model versus the collection language model. The term $\text{tf}(t_q, d)$ represents the term frequency of t_q in document d, and $|d|$ is the length of document d. Additionally, $\sum_{\bar{d} \in D} \text{tf}(t_q, \bar{d})$ is the total frequency of term t_q in the entire collection D, and $\sum_{\bar{d} \in D} |\bar{d}|$ is the total length of all documents in the collection D.

2. Dirichlet Prior Smoothing: It uses a Bayesian approach to smooth the document language model with the collection language model. The probability of a term t_q in document d is given by:

$$p(t_q|d) = \frac{\text{tf}(t_q, d) + \mu \frac{\sum_{\bar{d} \in D} \text{tf}(t_q, \bar{d})}{\sum_{\bar{d} \in D} |\bar{d}|}}{|d| + \mu} \tag{2.10}$$

Where: μ is the Dirichlet prior parameter, which controls the amount of smoothing [36, 37]. The term $\text{tf}(t_q, d)$ represents the term frequency of t_q in document d, and $|d|$ is the length of document d. Additionally, $\sum_{\bar{d} \in D} \text{tf}(t_q, \bar{d})$ is the total frequency of term t_q in the entire collection D, and $\sum_{\bar{d} \in D} |\bar{d}|$ is the total length of all documents in the collection D.

Without smoothing, a term that is missing from a document would have a probability of zero, which would highly affect retrieval scores. Smoothing reduces this issue by adjusting term probabilities based on collection-wide statistics. In Dirichlet smoothing, the parameter μ acts as a weight that determines how much influence the collection frequency should have. A higher values of μ leads to more reliance on the collection language model, making ranking more robust for shorter documents. This ensures better retrieval effectiveness by reducing the impact of data sparsity and improving score stability.

The Dirichlet smoothed model was chosen as the initial ranker in this study due to its ability to provide a stable and effective first-stage ranking before applying the NTLM reranking method.

2.2.2 Translation Language Model

The Translation Language Model, introduced by Berger and Lafferty [2], is an approach that addresses some of the limitations of traditional language modeling in Information

Retrieval (IR). This model redefines how query terms are generated from documents by incorporating a "translation" process, which allows for more flexible and semantically rich matching between queries and documents.
In the Translation Language Model, the query q is assumed to be generated through a translation process from the document d. This process is mathematically expressed as:

$$p(t_q|d) = \sum_{t_d \in d} p(t_q|t_d) \cdot p(t_d|d) \tag{2.11}$$

Where: t_q is a term in the query q, and t_d is a term in the document d. The term $p(t_q|t_d)$ represents the translation probability, which captures the likelihood that the query term t_q is a "translation" of the document term t_d. Additionally, $p(t_d|d)$ is the probability of the document term t_d in the document d, typically estimated using the maximum likelihood estimate (MLE):

$$p(t_d|d) = \frac{\text{tf}(t_d, d)}{|d|}, \tag{2.12}$$

where $\text{tf}(t_d, d)$ is the term frequency of t_d in document d, and $|d|$ is the length of the document.

The translation process allows the model to capture relationships between query terms and document terms that may not be identical but are semantically related. For example, the query term "automobile" might be translated from the document term "car,". This flexibility helps bridge the vocabulary gap between queries and documents, improving the effectiveness.

The key components of the Translation Language Model include:
Translation Probability $p(t_q|t_d)$: This component enables the model to bridge the vocabulary gap by allowing non-identical but semantically related terms to contribute to the relevance score. Berger and Lafferty proposed estimating $p(t_q|t_d)$ using query-document paired data. For instance, if a document containing the term "car" is frequently retrieved for queries containing the term "automobile," the translation probability $p(\text{automobile}|\text{car})$ would be high.

Document Term Probability $p(t_d|d)$: This component represents the importance of the document term t_d in the document d. It is typically estimated using the term frequency $\text{tf}(t_d, d)$ normalized by the document length $|d|$.

Summation Over Document Terms: The summation $\sum_{t_d \in d}$ ensures that all terms in the document contribute to the probability of generating the query term t_q. This allows the model to consider multiple document terms that might be related to the query term, even if they are not exact matches.

2.3 Neural Word Representation

The field of IR has evolved with the introduction of neural network models, which enhance retrieval effectiveness and efficiency beyond traditional heuristic and statistical methods [18, 24]. Neural IR models improve the understanding of complex queries and document relationships through learned representations [13]. This section explains how these neural methods influence document ranking, vector representation, and relevance estimation.

2.3.1 Vector Represenation

Vector representations are fundamental to both IR and machine learning [3, 38]. In IR, terms are the basic units for indexing and retrieval. This makes effective vector representations very important. These representations balance different levels of generalization—some treat each term as a unique entity, while others identify shared attributes among terms. The way vector spaces are defined determines how similarity between terms is measured, with some relying on fixed-size vocabularies and others avoiding such constraints [25].

There are two primary types of vector representations used in IR:

1. Local (or One-Hot) Representations
2. Distributed Representations

Local representations: Each term is assigned a unique binary vector, where only one position is "1" and the rest are "0." This approach treats terms as distinct entities with no inherent similarity. For example, in Figure 2.3, terms like "car," "bicycle," and "apple" are represented as independent vectors, capturing no semantic relationships.

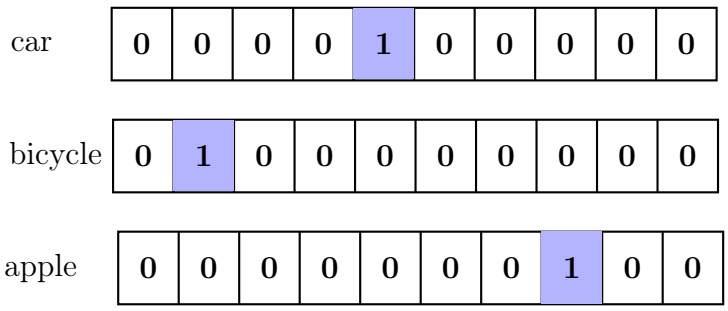

Figure 2.3: Local (One-Hot) Vector Representations

Distributed Representations: Each term is assigned a vector that captures its attributes, allowing terms to be compared based on shared features. For example, "car" might be closer in vector space to "bicycle" than to "apple" because both are modes of transportation. These vectors can be sparse or dense, with dense vectors (embeddings) learned from data. Neural models often use embeddings, where a term's meaning is captured by the combined activations of multiple neurons, providing a richer representation

than traditional methods.

Figure 2.4 illustrates distributed representations, where terms like "car," "bicycle," and "apple" are represented by vectors with multiple active dimensions, capturing semantic similarities based on shared attributes. For example, "car" and "bicycle" share dimensions like "has tire," while "apple" is associated with attributes like "fruit" and "edible."

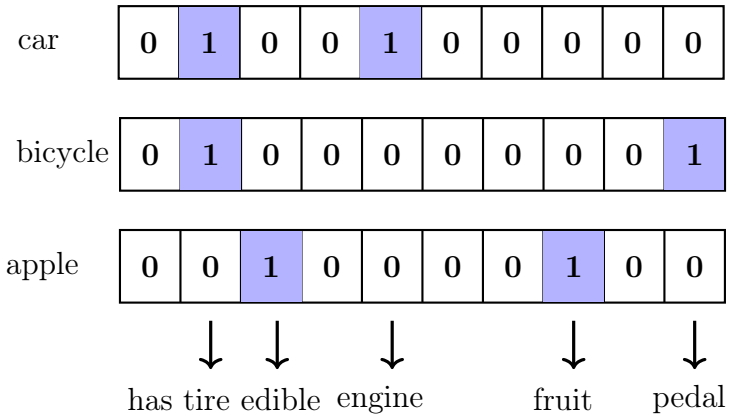

Figure 2.4: Distributed Vector Representations

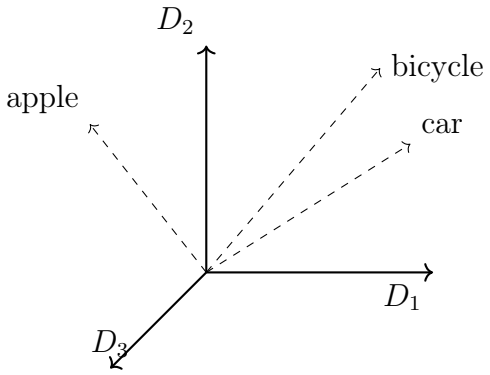

Figure 2.5: High Dimensional Vector Space Representation

Figure 2.5 shows how semantic relationships are captured in vector space. Terms like "car" and "bicycle" are closer in the D_1-D_2 plane, reflecting shared attributes, while "apple" is positioned along D_2-D_3, indicating fewer shared attributes. This highlights how similar terms are grouped based on common characteristics.

2.3.2 Dense Vector Representations Model

In natural language processing, two closely related dense vector representation models that are highly relevant to our thesis are the Skip-gram and Continuous Bag of Words (CBOW) models [22]. These popular techniques are used to learn word embeddings, which are dense vector representations of words. These embeddings capture the semantic relationships between words based on their context within a corpus.

1. Skip-gram Model The Skip-gram model, introduced by Mikolov [22], aims to predict the context words surrounding a given target word. For a given target word, the model uses its embedding to predict the probability of each word within a fixed-size context window around it. For example, in the sentence:

"Birds are flying over the field."

If "$flying$" is the target word, the Skip-gram model will learn to predict "$Birds$," "are," "$over$," "the" and "$field$" as the context words. This approach is particularly effective in capturing semantic similarities between words based on their usage in various contexts. Figure 2.6 below illustrates this concept.

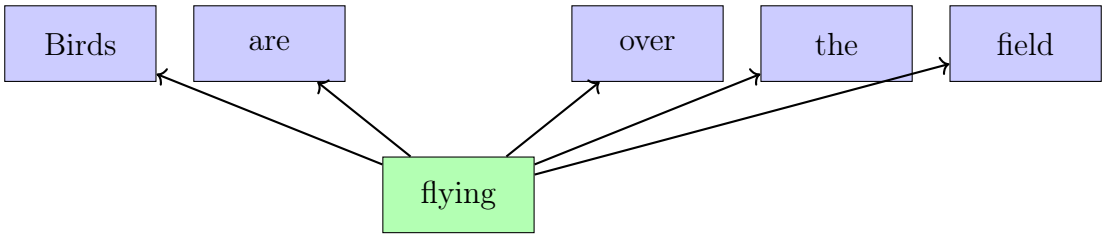

Figure 2.6: Skip-gram Model: The target word "flying" predicts its surrounding context words.

2. Continuous Bag of Words (CBOW) Model The CBOW model, also proposed by Mikolov [22] works in the reverse direction of Skip-gram. Instead of predicting context words from a target word, CBOW predicts a target word given its surrounding context. For a given context (i.e., a set of surrounding words), the model predicts the probability of the target word being within that context. Using the same example sentence:

"Birds are flying over the field,"

The CBOW model would use the context words "Birds," "are," "over," "the," and "field" to predict the target word "flying." This model is particularly effective for tasks where understanding the surrounding context of a word is important. Figure (2.7) shows this concept.

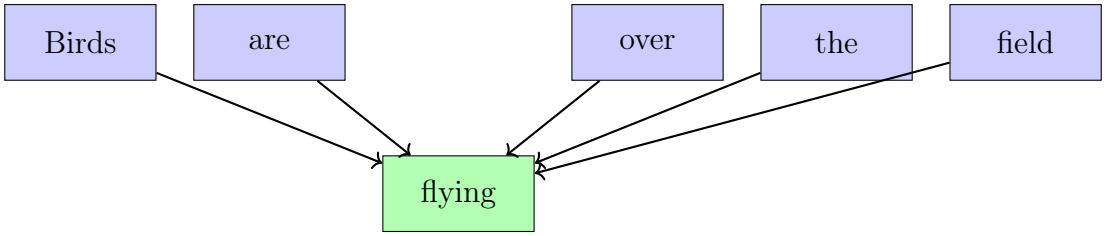

Figure 2.7: Continuous Bag of Words Model: The context words predict the target word "flying."

2.4 Transformer Models

The Transformer is a deep learning model architecture that revolutionized sequence-to-sequence tasks by replacing recurrent neural networks (RNNs) and convolutional neural networks (CNNs) with an attention mechanism. This mechanism allows the model to capture relationships between tokens in a sequence more effectively, making it highly efficient for tasks like machine translation, text generation, and information retrieval.

In this research, we focus on the initial part of the Transformer model—the input embedding layer. This layer is responsible for converting textual tokens into continuous vector representations that neural networks can process. These embeddings serve as learned representations, capturing the semantic and syntactic properties of each token in the input sequence. This process is particularly relevant to our work on the NTLM, which leverages Transformer input embeddings to improve retrieval effectiveness.

Token embeddings represent the fundamental mechanism for converting textual tokens into continuous vector representations that neural networks can process. In the Transformer architecture, these embeddings serve as learned representations that capture the semantic and syntactic properties of each token in the input sequence. The token embedding process begins with a learnable embedding matrix $E \in \mathbb{R}^{|V| \times d_{\text{model}}}$, where $|V|$represents the vocabulary size and d_{model} represents the embedding dimension. Each row in this matrix corresponds to a unique token in the vocabulary, and the values in this matrix are initialized randomly and then optimized during the training process through back-propagation. When a token is processed, its corresponding embedding vector is retrieved from this matrix through a simple lookup operation.

2.5 IR Evaluation

In this thesis, several standard metrics are used to evaluate the effectiveness of the retrieval models. Mean Average Precision (MAP) is used to assess the overall quality of the retrieved results [27], and Precision at 10 (P@10) focuses on the accuracy of the top 10 results [26]. Together, these metrics provide a comprehensive evaluation of model effectiveness.

Precision and recall Precision and recall both compute the fraction of relevant documents retrieved for a query q, but with respect to the total number of documents in

the retrieved set Rq and the total number of relevant documents in the collection D, respectively. Both metrics assume that the relevance labels are binary.

$$\text{Precision}_q = \frac{\sum_{(i,d)\in R_q} \text{rel}_q(d)}{|R_q|} \tag{2.13}$$

$$\text{Recall}_q = \frac{\sum_{(i,d)\in R_q} \text{rel}_q(d)}{\sum_{d\in D} \text{rel}_q(d)} \tag{2.14}$$

Mean average precision (MAP) The average precision [44] for a ranked list of documents R is given by:

$$\text{AveP}_q = \frac{\sum_{(i,d)\in R_q} \text{Precision}_{q,i} \times \text{rel}_q(d)}{\sum_{d\in D} \text{rel}_q(d)} \tag{2.15}$$

$$\text{MAP} = \frac{\text{AveP}_q}{\text{total No of Queries}} \tag{2.16}$$

where, $Precision_{q,i}$ is the precision computed at rank i for the query q. The average precision metric is generally used when relevance judgments are binary, although variants using graded judgments have also been proposed [31]. The mean of the average precision over all queries gives the MAP score for the whole set.

Mean reciprocal rank (MRR) Mean reciprocal rank [9] is also computed over binary relevance judgments. It is given as the reciprocal rank of the first relevant document averaged over all queries.

$$\text{RR}_q = \max_{h_i,d_i\in R_q} \frac{\text{rel}_q(d)}{i} \tag{2.17}$$

Normalized Discounted Cumulative Gain (nDCG) [15]. The Discounted Cumulative Gain (DCG) is a measure that incorporates both the relevance and position of documents in a ranked list. Unlike MAP, nDCG can handle graded relevance judgments naturally. For a ranked list R, DCG is computed as:

$$\text{DCG}_q = \sum_{i=1}^{|R_q|} \frac{2^{\text{rel}_q(d_i)} - 1}{\log_2(i+1)} \tag{2.18}$$

where $\text{rel}_q(d_i)$ is the relevance grade of document d_i for query q. To normalize DCG and obtain scores between 0 and 1, we divide by the ideal DCG (IDCG), which is computed by sorting documents by their relevance grades in descending order:

$$\text{nDCG}_q = \frac{\text{DCG}_q}{\text{IDCG}_q} \tag{2.19}$$

The final nDCG score for the system is computed by averaging over all queries.

Chapter 3

Experimental Setup

In this chapter, we provide an overview of the datasets used in this thesis for document retrieval and ranking tasks. The datasets are collected from: `BEIR` [16], `ir_datasets` [1] [20]. A total of 10 different datasets were selected to evaluate and compare the generalization ability of the retrieval models proposed in this work. The datasets encompass a wide range of domains, including both broad topical datasets, such as Wikipedia, and more specialized domains, like COVID-19 publications. Additionally, they cover various text types (e.g., news articles, research papers, finance), with varying query and document sizes.

The chosen datasets span a broad spectrum of sizes, from smaller collections (containing as few as 3.6k documents) to large-scale datasets (containing up to 3.2 million documents). They also differ in terms of query and document lengths, with average query lengths ranging from 3 to 13 words, and document lengths ranging from 11 to 292 words. This diversity allows for a thorough evaluation of model effectiveness across different settings, providing insights into the scalability and adaptability of retrieval models.

The datasets used in this thesis adhere to this experimental setup, ensuring the evaluation of retrieval models. Below, we describe the specific datasets used for evaluation.

3.1 Datasets

The following datasets are used in this thesis for retrieval and ranking tasks:

Associated Press (AP) Dataset: The Associated Press (AP) dataset, provided by the Linguistic Data Consortium (LDC), contains newswire articles from the Associated Press, dating back to the late 1980s. Specifically, we use articles from the AP88-89 collection (TREC disks 1 and 2), along with topics from TREC 1, 2, and 3 ad-hoc collections (topics 51–200). The dataset spans a wide array of topics, such as politics, business, sports, and societal issues, making it well-suited for general-purpose document retrieval research. Each document in the collection is uniquely identified and contains a body of text, providing a rich resource for exploring news-oriented information retrieval tasks.

[1] https://ir-datasets.com/

Wall Street Journal (WSJ) Dataset: The Wall Street Journal (WSJ) dataset consists of articles published in the Wall Street Journal during the late 1980s. It includes news articles from the WSJ87-92 collection (TREC disk 1), along with topics from TREC 1, 2, and 3 ad-hoc collections (topics 51–200). The dataset provides comprehensive coverage of business and financial matters, as well as a broad range of other topics such as technology, world affairs, and government policy. Similar to the AP dataset, the WSJ documents are structured with unique identifiers and text bodies, ensuring an organized and standardized collection suitable for document retrieval and ranking tasks.

DOTGOV Dataset: The DOTGOV dataset consists of webpages crawled from the ".gov" domain, specifically focusing on topics from the TREC 2002 (topics 551-600). The dataset aims to support research in information retrieval, particularly for government-related content. Judgments for relevance are binary, with documents classified as either relevant or irrelevant to the corresponding topics. This dataset offers a valuable resource for exploring document retrieval and ranking in the context of government websites.

Natural Questions (NQ) Dataset [19]: The Natural Questions (NQ) dataset is a large-scale question-answering dataset comprising real, anonymized, aggregated queries submitted to the Google search engine. In NQ, all relevance judgments are binary, with documents judged as either relevant (grade 1) or non-relevant, making it suitable for binary relevance evaluation in information retrieval tasks.

Finance Opinion Mining and Question Answering (FiQA) Dataset [21]: This dataset supports research in understanding financial text by enabling aspect-based sentiment analysis and opinion-based question answering. It includes English texts from diverse sources, such as microblogs, news, and reports, focusing on detecting target aspects and predicting sentiment scores (-1 to 1) or answering financial queries. Systems can either rank relevant documents or generate direct answers, with relevance judged as binary (relevant or not). This dataset enables in-depth exploration of domain-specific challenges, offering a detailed view of sentiment and opinion mining in financial contexts.

Quora Dataset [35]: The Quora dataset consists of pairs of questions from the Quora question-and-answer platform, with the task of determining whether the two questions are duplicates, i.e., if they have the same meaning. Quora is a popular site where users post questions and provide answers, and the best responses are up-voted. Judgments for this dataset are binary, classified as either relevant or irrelevant.

SciFact Dataset [41]: SciFact introduces the task of scientific claim verification, where the goal is to identify abstracts from the research literature that either support or refute a given scientific claim, along with providing rationales for each decision. The SciFact dataset contains 1.4K expert-written scientific claims paired with evidence-rich abstracts, annotated with labels and rationales.

Nutrition Facts (NF) Dataset [5]: The NF corpus is a dataset designed for learning-to-rank tasks within the medical domain. It includes thousands of full-text queries connected to numerous research articles. The queries originate from health-related topics

written in plain English, sourced from the non-commercial website NutritionFacts.org. Relevance assessments are categorized into three levels, derived from both direct and indirect connections between the queries and PubMed research articles, providing a valuable resource for evaluating medical information retrieval systems.

TREC-COVID Dataset [40]: is an IR dataset in the biomedical domain consisting of questions about Coronavirus and scientific articles as document collection. We use the question from the query set and the documents from the COVID-19 Open Research Dataset. Relevance judgments in TREC-COVID are categorized into three grades: relevant, partially relevant, and non-relevant. For evaluation purposes, we consider all three relevance levels.

Webis-Touché 2020 Dataset [4]: Webis-Touché 2020 is an argument retrieval dataset comprising arguments collected from debate websites. The queries are presented as direct questions and relevance judgments are divided into three levels: relevant, partially relevant, and non-relevant. For evaluation, all three relevance levels are considered.

MS MARCO-Document TREC 2020 DL Dataset [10]: The MS MARCO TREC Deep Learning 2020 dataset is a large-scale information retrieval dataset designed to support research on document and passage ranking using deep learning methods. The dataset supports document ranking, enabling the development and evaluation of retrieval models under a large training data regime. Relevance judgments are provided on a four-point scale: perfectly relevant, highly relevant, relevant, and irrelevant, offering good evaluation criteria for ranking models.

Table 3.1: Statistics and Characteristics of the Datasets

Dataset	Domain	Relevancy	#Query	#Corpus	Avg Word Length	
					Query	Document
AP	News	Binary	150	242K	5.2	285.1
WSJ	Finance	Binary	150	173K	5.2	283.5
DOTGOV	Governmental	Binary	50	1.24M	3.3	72.34
NQ	Wikipedia	Binary	3452	2.6M	9.16	78.88
FiQA	Finance	Binary	500	57.6K	10.77	132.32
Quora	Quora	Binary	300	522.9K	9.53	11.44
SciFact	Scientific	Binary	300	5.1k	12.37	213.63
NFCorpus	Biomedical	3-level	324	3.63K	3.30	232.26
TREC-COVID	Biomedical	3-level	50	171K	10.60	160.77
Webis-Touché-2020-v2	Mics	3-level	49	382K	6.55	292.37
MS MARCO-Document	Document Retrieval	4-level	200	3.2M	5.8	234.1

3.2 Retrieval Setup

The retrieval setup for the NTLM experiment is designed to leverage advanced embedding techniques and efficient retrieval models. The experiment uses the PyTerrier platform for indexing, ranking, and evaluation. PyTerrier is chosen for its flexibility and efficiency in handling large-scale retrieval tasks. Additionally, it integrates the smoothing parameter easily into the model and has futures to save the initial ranked result with their corresponding documents to use for the reranking task.

Static word embeddings are used to enhance document and query representations. We use the Word2Vec algorithm, implemented via the `gensim` library, to generate these embeddings. Word2Vec offers two architectures: Skip-gram and CBOW. Both architectures are used to assess their impact on retrieval effectiveness.

We also extract input embeddings from pre-trained transformer models (e.g., monoBERT, GPT-2, RoBERTa) by accessing their token embedding layers. For each model, we load the tokenizer and model architecture from Hugging Face, then retrieve the word embeddings layer. These embeddings, which map tokens to high-dimensional vector representations, are converted into `NumPy` arrays for efficient manipulation and stored in a dictionary mapping tokens to their corresponding embeddings. This approach ensures consistent access to pre-trained semantic representations, which are integrated into our retrieval model.

Additionally, WordLlama leverages the token embedding layer of large language models like LLaMA to extract dense vector representations of words and subwords. For our implementation, we use the `l3_supercat_1024.safetensors` file, containing a 1024-dimensional embedding matrix for 128,256 tokens, and the `l3_supercat_tokenizer_config.json` file, which defines the tokenizer configuration. These files, sourced from the WordLlama L3 Supercat repository on Hugging Face [2], enable rich semantic representations of text. The embeddings are stored in an optimized `safetensors` format for efficient access and integrated into a model to enhance retrieval effectiveness.

[2]https://huggingface.co/dleemiller/wordllama-l3-supercat/tree/main

Chapter 4

Methodology

In this chapter, we describe the methods used to implement the Dirichlet smoothed model and the NTLM. The Dirichlet model serves as the initial ranker, representing a probabilistic method, while the NTLM is implemented using two types of embeddings: Word2Vec and input embeddings from transformer models. We will detail the process from data preparation to the final implementation, highlighting how each model is structured and integrated to assess its effectiveness in document retrieval and ranking.

4.1 Dirichlet Language Model

This study implemented the Dirichlet Smoothed Language Model as the initial ranker probabilistic retrieval method. This model, grounded in language modeling principles, estimates the $p(w|d)$ by mixing the maximum likelihood estimation, $p_{ml}(w|d)$, with the collection background probability $p(w|C)$. Dirichlet smoothing is applied to address unseen terms in documents, ensuring that the probability of observing a query term is never zero, even when the term does not appear in a particular document, as we see in Equation 2.10

Building on the foundational work of Zhai and Lafferty [42] on Dirichlet smoothing language models, we use PyTerrier[1] to implement and evaluate the Dirichlet smoothing language model across multiple datasets. PyTerrier provides a flexible and efficient framework for indexing, retrieval, and evaluation, making it ideal for our experiments. After preprocessing and indexing the datasets, we configured the Dirichlet Language Model with the appropriate smoothing parameter μ, which plays an important role in balancing document-specific term frequencies with collection-wide term probabilities.

The smoothing parameter μ was fine-tuned for each dataset to optimize retrieval effectiveness. For example, the optimal values of μ were determined to be 600 for the AP88-89 collection and 900 for the WSJ87-92 collection. These values were chosen based on empirical evaluation, ensuring that the model effectively addresses variations in document length and term distributions. The retrieval process involved querying the indexed documents using the Dirichlet model, which computes relevance scores for each document based on the smoothed probability estimates. The results were then ranked and returned in descending order of relevance, providing a ranked list of documents for each query.

[1] `https://pyterrier.readthedocs.io/en/latest/installation.html`

This initial ranked list serves as the input for a subsequent reranking task using the NTLM.

4.2 Static Word Embeddings Implementation

In this study, we use static word embeddings to enhance document and query representation for the NTLM. The goal is to evaluate the effectiveness of static embeddings generated from different datasets and models, focusing on their impact on retrieval and ranking tasks.
We leverage the `Word2Vec` algorithm, implemented using the `gensim` library, to generate static word embeddings. Word2Vec provides two main architectures: Skip-gram and CBOW. By using these two models, we aim to investigate how each affects the quality of the embeddings and, in turn, their impact on retrieval effectiveness.

Table 4.1: Datasets and Model Parameters used during training to create Word2Vec model.

Dataset	**AP**	**WSJ**	**DOTGOV**
Dataset Size:	0.72 GB	0.52 GB	19 GB
Number of documents:	243 k	173 k	1.25 M
Number of tokens:	69 M	49 M	903 M
Number of terms:	301 k	176 k	2.90 M
Embedding Dimension:	300	300	300
Window Size:	5	5	5
Min Word Frequency:	2	2	2
Negative Samples:	15	15	15
Epochs:	50	50	30

The Word2Vec models are trained on different datasets, each representing distinct textual domains. This allows us to assess how domain-specific training influences the embeddings and their use in various tasks. The datasets used for training include:AP88-89 and WSJ87-92 datasets. Each dataset is tokenized, preprocessed, and used to train both the Skip-gram and CBOW models, yielding different sets of embeddings for comparison. The training parameters, such as the context window size, embedding dimension, and minimum word frequency, were fine-tuned based on preliminary experiments.

In addition to training custom word embeddings, we incorporate the widely used `Google News-vectors-negative300` embeddings from the NLPL repository [2]. These embeddings were pre-trained on the Google News dataset, consisting of approximately 100 billion words. The model uses the Skip-gram architecture with 300-dimensional vectors, capturing a vast range of word relationships from general news articles.
For domain-specific applications, we also use an Oil and Gas corpus embedding model, adapted to industry-specific language. This model was trained using the CBOW architecture, producing 400-dimensional embeddings. The CBOW model is known for its efficiency in representing frequent co-occurrences, which is advantageous for capturing terms and relationships unique to the oil and gas sector.

[2] http://vectors.nlpl.eu/repository/

Table 4.2: static word Embedding Models Taken from NLPL Repository

Corpus	Model	Vocabulary Size	Dimensions	Size
Google News 2013	Skip-gram	3,000,000	300	3.3 GB
Oil and Gas Corpus	CBOW	285,055	400	450 MB

4.3 Input Embeddings from Transformers Model

In this research, we also leverage pre-trained transformer models to obtain rich word representations. The process begins with accessing the word embeddings layer, which is the first layer of these transformer architectures. This layer serves as an advanced lookup table where each token in the model's vocabulary is mapped to a high-dimensional vector representation. These vectors are not randomly initialized but have been carefully optimized during the pre-training phase on large text corpora, enabling them to capture semantic relationships between words.

To create these embeddings, we use a systematic extraction process from the word embeddings layer. Each token in the model's pre-trained vocabulary has an associated embedding vector within this layer, represented as a dense vector of floating-point values. By extracting the weights from the embedding layer, we obtain an embedding matrix where each row corresponds to the embedding of a unique token in the vocabulary. For example, in BERT-base, this matrix has dimensions of 30,522 $\times$ 768, where 30,522 represents the vocabulary size and 768 is the embedding dimension. The entire matrix encapsulates the model's learned knowledge about word meanings and relationships.

The extraction process fully preserves these pre-trained representations, retaining the semantic information learned during the model's training phase. These embeddings are then stored in a structured format, typically as memory-efficient numpy arrays, ensuring fast access and retrieval during the cosine similarity computation.

Table 4.3: Input Embedding Dimensions and Vocabulary Sizes of Different Models

Model	Embedding Dimension	Vocabulary Size
BERT-base	768	30,522
GPT-2	768	50,257
RoBERTa-base	768	50,265
MonoBERT-large-msmarco	1024	30,522

In this study, we evaluate how different models with varying embedding dimensions influence retrieval and ranking. We use input embeddings from BERT-base, RoBERTa-base, GPT-2, and MonoBERT (`castorini/monobert-large-msmarco`). MonoBERT is a variant of BERT, fine-tuned for passage ranking tasks such as MS MARCO documents. This setup allows us to assess the impact of embedding size and model fine-tuning on ranking performance. Based on these input embeddings, we show their impact on the ranking task to understand how model selection influences retrieval effectiveness.

4.4 Word Translation Using Word Embeddings

Before first directly analyzing the results, it is important to understand how word translation is analyzed using different embedding models, as this defines how words are mapped across languages in an embedding space. By examining word-to-word translations, we can uncover semantic relationships and ensure that embeddings effectively capture linguistic meaning and improve translation effectiveness.

This section outlines the methodology for implementing a word translation method using various word embedding models, including Word2Vec, BERT, RoBERTa, and MonoBERT. The core approach involves leveraging the semantic representations captured by these embeddings to identify translation candidates for a given source word. While the overall method is consistent across models, each embedding technique has unique characteristics that influence its implementation and effectiveness. The methodology is divided into the following components: (1) initialization of the models and extraction of word embeddings, (2) computation of word similarities, (3) calculation of translation probabilities, and (4) generation of translation candidates. A comparison of the techniques used in each model is also provided.

To measure semantic similarity, cosine similarity is used across all models. Values closer to 1 indicate higher similarity. For a given source word, its embedding is obtained by tokenizing the word and extracting the embedding of the first token, ensuring consistency across different tokenization methods. While Word2Vec does not require tokenization, BERT and MonoBERT use WordPiece tokenization, GPT2 uses byte pair encoding (BPE), and RoBERTa uses byte-level BPE tokenization.

The translation probability $p_t(w|u)$ of a target word w given a source word u is calculated using a softmax function with temperature scaling. This involves three steps: computing the cosine similarity between the source and target embeddings, scaling the similarity by a temperature parameter (typically around 0.9), and normalizing the probabilities using the top-k nearest neighbors. Finally, translation candidates are generated by identifying the top-k nearest neighbors, calculating their probabilities, and retaining the top similar words for the given source word based on their computed probability.

The diagram 4.1 shows how different words in a document can translate into a given query term with varying probabilities. In the diagram, the nodes represent words, and the edges denote translation probabilities. For example, if “Forest” is a query term, the surrounding words such as “Jungle,” “Rainforest,” “Woodland,” and “Tree” represent document terms that might contribute to generating this query. Each connection in the diagram is weighted by a translation probability $p_t(w|u)$, where u is a document term and w is the query term.

Words with a stronger semantic relationship, such as “Jungle” and “Rainforest,” typically have higher translation probabilities, as they are more likely to be associated with the query term “Forest.” In contrast, less directly related terms like “Tree” and “Woodland” may have lower probabilities, reflecting their weaker semantic connection to “Forest.” The self-loop on “Forest” indicates that exact matches are also considered, reinforcing the importance of direct term occurrences. This self-translation probability $p(\text{Forest}|\text{Forest})$

accounts for cases where the query term is generated directly from itself, independent of other document terms. In most cases, this probability is higher, as exact matches are typically more relevant than translations from other terms.

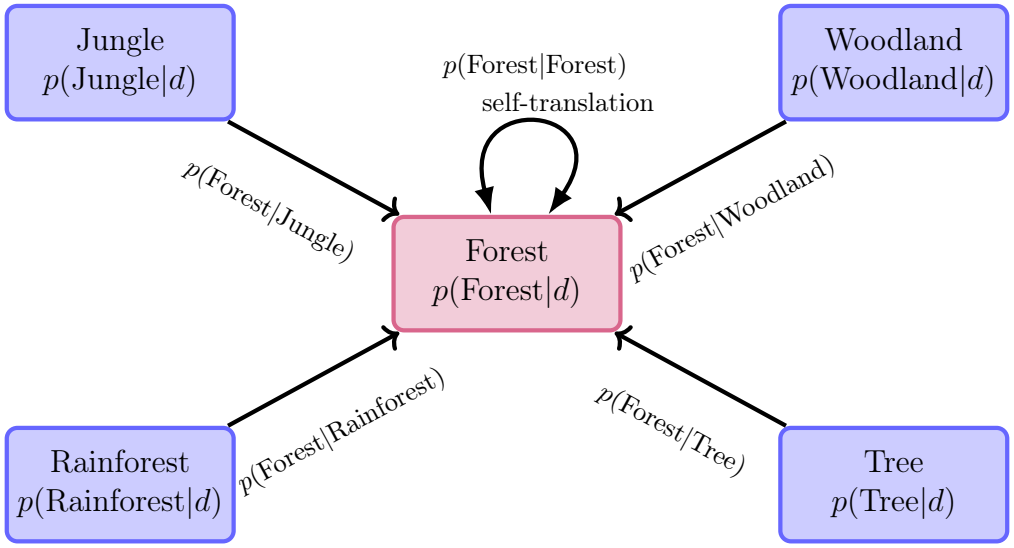

Figure 4.1: Probabilistic translation model showing how document terms (e.g., "Jungle," "Rainforest") contribute to generating a query term ("Forest"). Edges represent translation probabilities $p_t(w|u)$, and the self-loop denotes the self-translation probability $p(\text{Forest}|\text{Forest})$.

4.5 Neural Translation Language Model (NTLM)

The NTLM method from previous work by Zuccon [45] builds on the foundational work of Berger and Lafferty [2], who introduced a method to estimate $P(w|d)$ by framing information retrieval as a form of machine translation. In their approach, the likelihood of generating a query depends on translating document terms into corresponding query terms. The NTLM extends this idea by incorporating advanced techniques such as word embeddings and cosine similarity to improve the estimation of translation probabilities. While both approaches treat query generation as a translation process, the NTLM leverages word embedding to improve the estimation of translation probabilities $P_t(w|u)$ as shown in Equation 2.11.

As Karimzadehgan and Zhai [17] noted, this translation probability allows for the incorporation of semantic relationships between terms, even for terms that do not explicitly co-occur in the document, thus providing a form of semantic smoothing for $P(w|d)$.
A key challenge in translation language models is accurately estimating $P_t(w|u)$, the probability of translating u into w. To address this, word embeddings can be used, using cosine similarity as a proxy for $P(u|w)$:

$$P_{cos}(u|w) = \frac{\cos(u, w)}{\sum_{u' \in V} \cos(u', w)}, \tag{4.1}$$

where $\cos(u, w)$ is the cosine similarity between the vector representations of the word

u and the query term W, and the denominator normalizes this value into a probability distribution over all possible translations.

4.6 Ranking Documents using the NTLM

In the language modeling approach, documents are ranked by computing the log-likelihood of the query given the document [43]. In this study, we used the following equation as the foundational formula for document ranking using the NTLM:
The log-likelihood Equation 4.2 below represents the probability of a query given a document. This measure is central to document ranking by assessing which documents are more likely to generate a specific query. Documents with higher log-likelihood values for a query are ranked higher, as they are considered more relevant to the query.

$$\log p(q|d) = \sum_{i:c(q_i;d)>0} \log \left(\frac{p(q_i|d)}{\alpha_d \cdot p(q_i|C)} \right) + n \log \alpha_d + \sum_{i=1}^{n} \log P(q_i|C) \tag{4.2}$$

In this equation, $p_s(q_i|d)$ is the smoothed probability of query term q_i given document d, α_d is the document-dependent smoothing parameter, $p(q_i|C)$ is the probability of query term q_i given the collection C, and n represents the number of query terms (or the length of the query). The last term in the equation is document-independent and can be ignored for ranking purposes.

Translation Model for Query Likelihood: The query generation process is modeled as a translation from document terms to query terms. The translation probability $p_t(w|d)$ is calculated as:

$$p_t(w|d) = \sum_{u \in d} p_t(w|u) p(u|d) \tag{4.3}$$

In this equation, $p_t(w|u)$ represents the probability of translating a candidate document term u into a target query term w and can be expressed as:

$$p_{\mathrm{t}}(w \mid u) = \frac{\cos(w, u)}{\sum_{u' \in V} \cos(w, u')} \tag{4.4}$$

where $\cos(u, w)$ is the cosine similarity between the vector representations of words w and u, V is the vocabulary. The denominator normalizes the cosine values to produce a valid probability distribution over all possible translations.
$p(u|d)$ is the probability of term u occurring in document d. The probability $p(u|d)$ is given by:

$$p(u|d) = \frac{\mathrm{tf}(u, d)}{\sum_{v \in d} \mathrm{tf}(v, d)} \tag{4.5}$$

where $\mathrm{tf}(u, d)$ is the term frequency of u in document d and $\sum_{v \in d} \mathrm{tf}(v, d)$ is the total number of terms in document d.
Similarly, the probability of a query term q_i occurring in the collection C is calculated as:

$$p(q_i|C) = \frac{\mathrm{cf}(q_i, C)}{\sum_{v \in C} \mathrm{cf}(v, C)} \tag{4.6}$$

where $\text{cf}(q_i, C)$ is the collection frequency of query term q_i in the collection C, and $\sum_{v \in C} \text{cf}(v, C)$ is the total number of terms in the collection.
Most smoothing methods make use of two distributions, a model ps(w | d) used for "seen" words that occur in the document, and a model pu(w | d) for "unseen" words that do not. The probability of a query q can be written in terms Of these models as follows, where $c(w; d)$ denotes the count of word w in d:

$$\log p(q|d) = \sum_i \log p(q_i|d). \tag{4.7}$$

$$\log p(q|d) = \sum_{i:c(q_i;d)>0} \log p_s(q_i|d) + \sum_{i:c(q_i;d)=0} \log p_u(q_i|d). \tag{4.8}$$

$$\log p(q|d) = \sum_{i:c(q_i;d)>0} \log \left(\frac{p_s(q_i|d)}{p_u(q_i|d)} \right) + \sum_i \log p_u(q_i|d). \tag{4.9}$$

The probability of an unseen word is typically taken as being proportional to the general frequency of the word, e.g., as computed using the document collection. So, let us assume that $p_u(q_i|d) = \alpha_d p(q_i|C)$, where α_d is a document-dependent constant and $p(q_i|C)$ is the collection language model. Now we have:

$$\log p(q|d) = \sum_{i:c(q_i;d)>0} \log \left(\frac{p_s(q_i|d)}{\alpha_d p(q_i|C)} \right) + n \log \alpha_d + \sum_i \log p(q_i|C). \tag{4.10}$$

where n is the length of the query. Note that the last term on the right-hand side is independent of the document d, and thus can be ignored in ranking.
Combining the translation model and collection probabilities, the final log-likelihood for document ranking is:

$$\log p(q|d) = \sum_{i:c(q_i;d)>0} \log \left(\frac{\sum_{u \in d} p_t(q_i|u) p(u|d)}{\alpha_d p(q_i|C)} \right) + n \log \alpha_d. \tag{4.11}$$

4.7 WordLlama for Reranking

In addition to Word2Vec and input embedding from transformer models, we incorporate the WordLlama model proposed by Miller [23]. WordLlama recycles components from large language models (LLMs) to create efficient and compact word representations, similar to Word2Vec or GloVe. By extracting the token embedding codebook from state-of-the-art LLMs (e.g., LLaMA 2, LLaMA 3 70B), WordLlama trains a small, context-less model within a general-purpose embedding framework. This approach results in a lightweight model that performs well on tasks such as text similarity, ranking, and fuzzy deduplication. Despite its compact size—e.g., a 16MB default model with 256 dimensions—it provides a highly efficient alternative to larger models like Word2Vec.

Large language models like LLaMA or BERT have extensive token embedding matrices due to their large vocabularies. To create efficient word representations, WordLlama borrows the embedding matrix (codebook) from these large models by extracting the token embeddings layer. The process is outlined as follows:

Token Embedding Layer: Large LLMs have a dedicated layer usually the first layer in the model where each token in the vocabulary is assigned a unique embedding vector. This layer is trained to capture the meaning of each token in the context of the model's training data.

Extracting Embeddings: WordLlama's method accesses this token embedding layer to extract the full matrix. This matrix can be saved in a format such as safetensors for optimized storage. For example, in LLaMA, this matrix may be found in the model's first safetensors file, e.g., `l2_supercat_256.safetensors`.

Saving the Codebook: Once extracted, WordLlama can use this codebook in a lighter model. In some cases, the codebook embeddings from multiple models (like `l2_supercat .safetensors` and `l3_supercat.safetensors`) are concatenated to improve diversity and generalization, making the embeddings useful across a variety of tasks. In our implementation, We use the best model suggested, which is `l3_supercat_1024.safetensors` with around 128256 vocabulary words and 1024 dimensions.

We propose that embeddings from WordLlama have the potential to enhance the NTLM effectiveness. This proposition is based on the idea that WordLlama's lightweight yet high-quality embeddings, which draw on LLaMA token representations, could serve as effective word-level features for the NTLM. If WordLlama's embeddings capture semantic relationships more efficiently, as suggested by its success on Massive Text Embedding Benchmark MTEB[3].

[3] `https://github.com/dleemiller/WordLlama?tab=readme-ov-file#quick-start`

Chapter 5

Experimental Results

5.1 Comparative Analysis of the NTLM

The Dirichlet Language Model (DLM) serves as an initial ranker model, providing the initial rankings for the experiments. This probabilistic model addresses the challenge of term sparsity in document retrieval by using Dirichlet smoothing, which balances the influence of document-specific term frequencies and collection-wide statistics. The smoothing parameter μ is key in this balancing act: when μ is small, the model relies more on the frequency of terms within each document, while larger μ values shift the focus toward the overall collection statistics.

To create a strong initial ranker, we evaluated the Dirichlet model's effectiveness across multiple test collections, with a primary focus on optimizing the smoothing parameter μ to maximize retrieval effectiveness. Specifically, we tuned μ and observed its impact on MAP and nDCG scores for the top 100 documents. For datasets with binary relevance judgments, we used MAP, while nDCG was applied for datasets with graded relevance judgments, such as TREC-COVID, NFCorpus, and Webis-Touché2020-v2.

When ranking documents, the model aims to assess the likelihood that terms from a query will appear in a given document. However, relying only on term frequencies within each document can cause issues. If a term is frequent in a document, it may indicate relevance, but if the document is short or lacks a particular query term, this reliance could lead to a zero probability for relevant but unmatched documents. To mitigate this, collection-wide statistics provide a smoothing effect, ensuring that even terms absent from a document have a small, non-zero probability of being relevant.
The μ parameter in the Dirichlet model adjusts the balance between document-specific term frequencies and collection-wide statistics. With a high μ, the model places more weight on collection-wide term frequencies, which is useful for capturing broader context. With a low μ, the model depends more heavily on document-specific term frequencies, focusing more on the terms that actually appear within each document.

Figure 5.1, 5.2, 5.3 below shows the relationship between various μ values and MAP scores across different datasets. The plot shows MAP scores obtained at different μ values, ranging from 0 to 2000 in increments of 300. As shown, MAP increases quickly as μ moves from 0 to 300. Beyond this range, MAP remains steady for almost all datasets, except for DOTGOV, Quora, FiQA, and TREC-Covid datasets, it gradually decreases as μ continues to increase beyond 300. Our analysis indicates that the model performs optimally within a μ different range between 300-900 for datasets. Refer to Table 5.1 below for a detailed comparison.

In particular, the MAP scores are very low when $\mu = 0$. In this scenario, the Dirichlet Language Model behaves like a maximum likelihood estimate (MLE), relying exclusively on document-specific term frequencies. which proves problematic given our average query length of 4-10 words as shown in Table 3.1. Without smoothing, any document lacking one or more query terms is assigned a zero probability, regardless of their potential relevance. This leads to significant limitations in the retrieval process. Relevant documents lacking exact query terms are completely excluded from the results, and the model fails to recognize related terms or broader context. These constraints explain the consistently low MAP scores observed at $\mu = 0$ across all datasets. This highlights the importance of appropriate smoothing for effective retrieval.

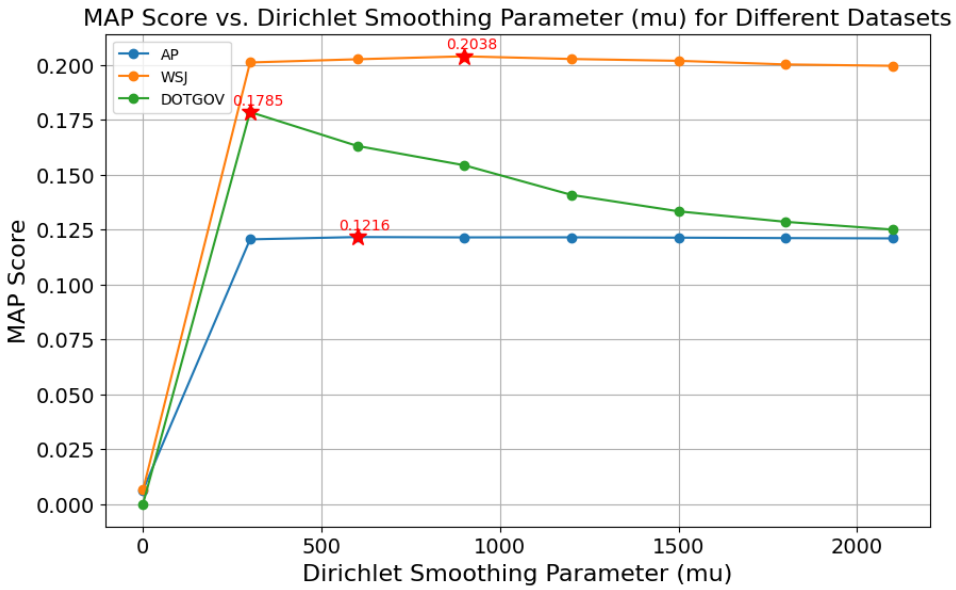

Figure 5.1: Mean Average Precision (MAP) scores for the top 100 documents at various μ values across AP, WSJ, and DOTGOV datasets.

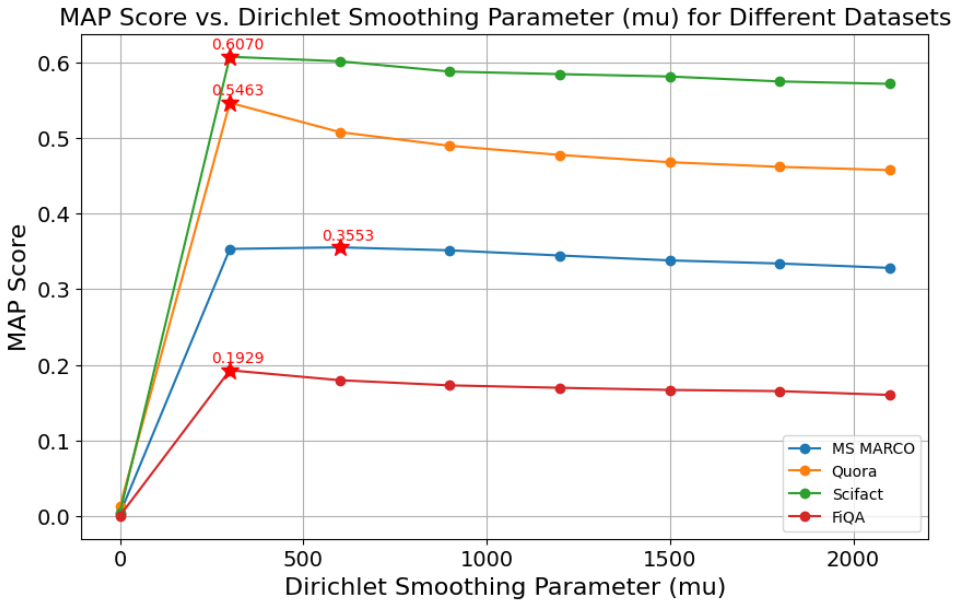

Figure 5.2: Mean Average Precision (MAP) scores for the top 100 documents at various μ values across MS-MARCO, Quora, Scifact, and FiQA datasets.

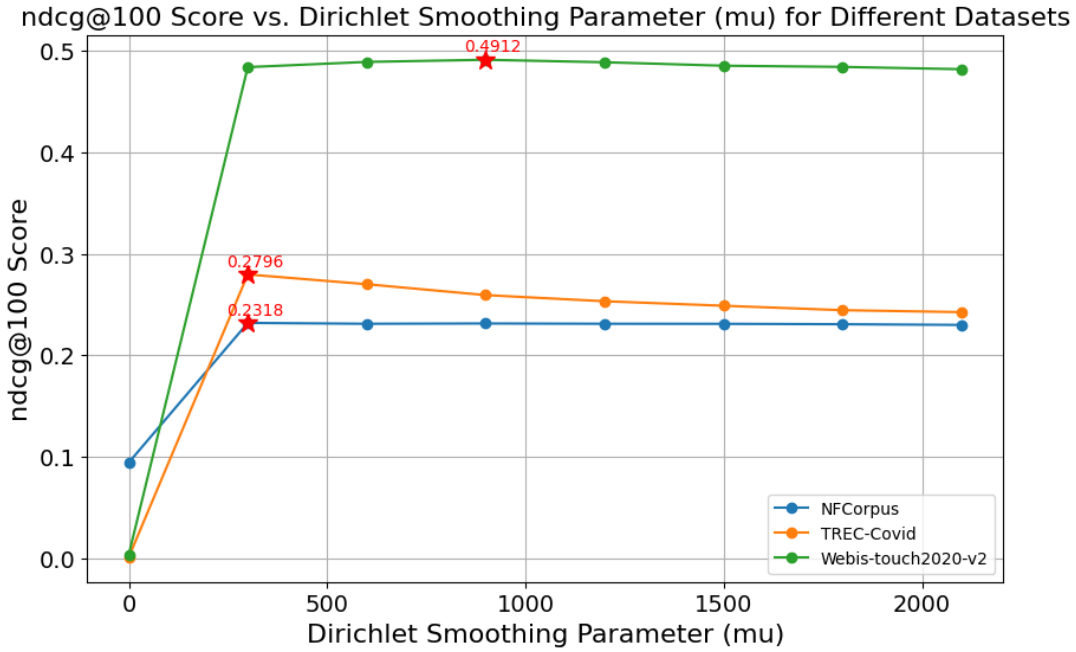

Figure 5.3: ndcg@100 scores for the top 100 documents at various μ values across NF-Corpus, TREC-Covid, and Webis-touch2020 datasets.

Table 5.1: Optimal μ values for maximizing MAP@100 and nDCG@100 in the Dirichlet Smoothed Model across different datasets. This table presents the best-performing μ parameter settings that achieve the highest retrieval effectiveness for each dataset.

Dataset	μ Parameter	Max MAP@100	Max nDCG@100
AP	600	0.12	
WSJ	900	0.20	
DOTGOV	300	0.17	
NQ	300	0.19	
MS MARCO	600	0.35	
FiQA	300	0.19	
Quora	300	0.60	
SciFact	300	0.60	
NFCorpus	300	-	0.23
TREC-COVID	300	-	0.27
webis-touche2020-v2	900	-	0.49

5.2 Results and Analysis of NTLM Retrieval

In information retrieval, particularly with language models, the goal is to accurately estimate the probability of a query given a document, $p(q|d)$. This probability depends on both document-specific term probabilities and collection-wide term frequencies. As shown in Eq.(4.10), the alpha parameter (α_d) balances these components, ensuring effective handling of both "seen" and "unseen" words. In the NTLM model's log-likelihood equation Eq.(4.11), α_d adjusts the influence of the collection language model $p(q_i|C)$ relative to document-specific term probabilities. Fine-tuning α_d is important because it balances specificity and generality: a higher α_d emphasizes document-specific terms, while a lower α_d leverages collection-wide frequencies. It also smooths probabilities for unseen words and normalizes document length, preventing bias towards longer or shorter documents. This ensures optimal retrieval effectiveness adapted to each dataset's characteristics.

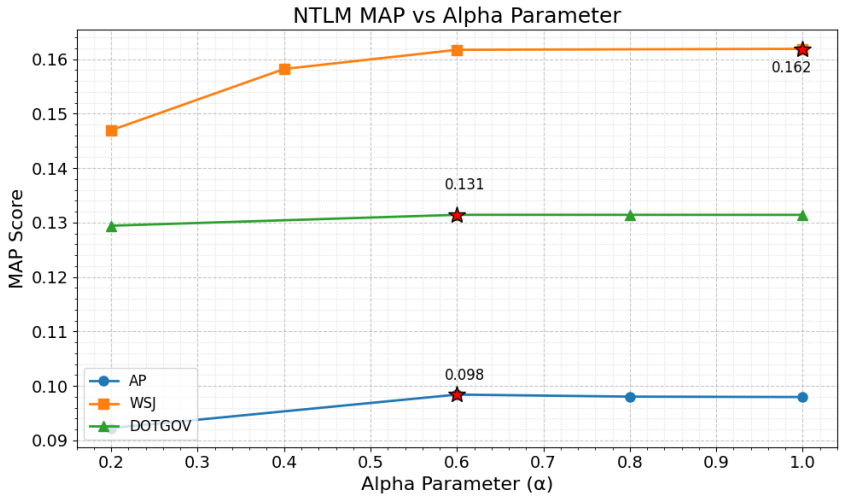

Figure 5.4: NTLM Mean Average Precision (MAP) scores for the top 100 documents at various α values across AP, WSJ, DOTGOV datasets.

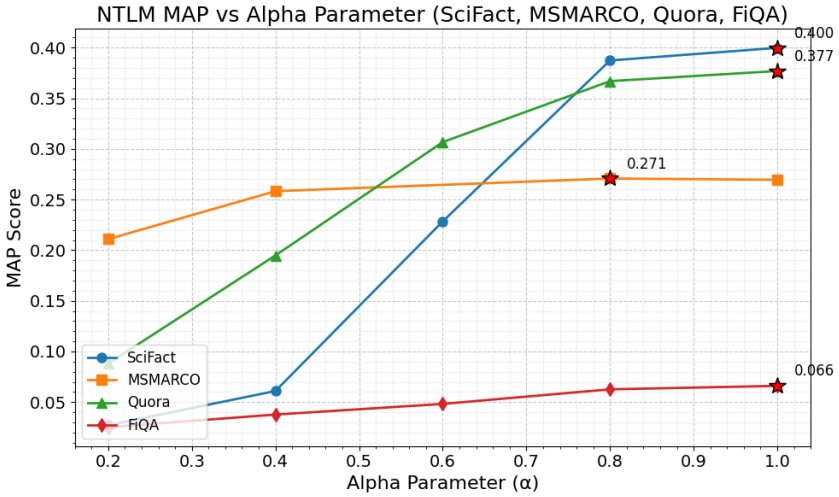

Figure 5.5: NTLM Mean Average Precision (MAP) scores for the top 100 documents at various α values across SciFact, MSMARCO, Quara and FiQA datasets.

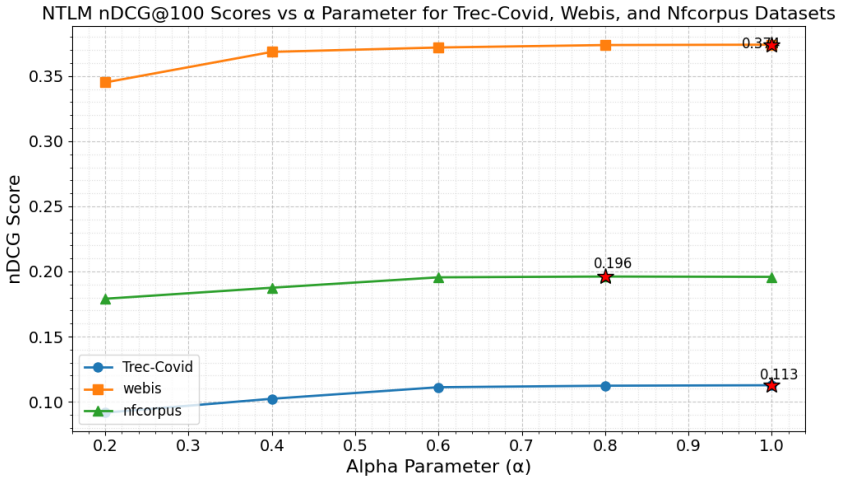

Figure 5.6: NTLM nDCG scores for the top 100 documents at various α values across TREC-Covid-19, Webis-Touche-2020-V2, NFCorpus datasets.

The experimental results, as illustrated in the plot graphs Figure 5.4, 5.5, 5.6 and summarized in Table 5.2, reveal insights about the effectiveness of the NTLM model across various datasets. The optimal α values vary considerably, indicating that the balance between document-specific and collection-wide term probabilities must be tailored to each dataset. For instance, datasets like Quora, SciFact, and webis-touche2020-v2 achieve their best effectiveness at $\alpha = 1.0$, suggesting that these datasets benefit from a stronger reliance on document-specific term probabilities. In contrast, datasets such as AP and DOTGOV perform optimally at $\alpha = 0.4$, highlighting the need for a more balanced approach. The MS MARCO dataset achieves a MAP@100 of 0.272 at $\alpha = 0.6$, demonstrating the importance of moderate parameter tuning for large-scale datasets.

These variations in precision highlight the importance of fine-tuning α to optimize retrieval effectiveness based on the unique characteristics of each dataset. The analysis, supported by the plot graphs and summarized table, emphasizes the role of the α parameter in achieving optimal retrieval effectiveness across diverse datasets.

Table 5.2: Optimal α Parameter and Max MAP@100 and nDCG@100 using the NTLM model for Different Datasets

Dataset	**α Parameter**	**Max MAP@100**	**Max nDCG@100**
AP	0.4	0.100	
WSJ	0.8	0.162	
DOTGOV	0.4	0.132	
MS MARCO	0.6	0.272	
FiQA	1.0	0.066	
Quora	1.0	0.377	
SciFact	1.0	0.40	
NFCorpus	0.8	-	0.196
TREC-COVID	1.0	-	0.113
webis-touche2020-v2	1.0	-	0.374

5.3 Comparison of the NTLM with Previous Related Works

In this section, we compare the effectiveness of the NTLM with the results reported in previous work by Zuccon [45]. The comparison is based on MAP and P@10 scores across three datasets: AP88-89, WSJ87-92, and DOTGOV for the top 1000 documents. Both works use the Dirichlet smoothed model as an initial ranker and we use the same parameters for a fair comparison.
Table 5.3 summarizes the results from the previous work and our current implementation. The MAP and P@10 scores are calculated for the top 1000 documents.

Table 5.3: Comparison of MAP and P@10 scores for top 1000 documents between previous work and our results. Bold values indicate the highest scores achieved for each metric (MAP or P@10) across models for a given dataset.

Method	**AP88-89** ($\mu = 1,000$)		**WSJ87-92** ($\mu = 1,500$)		**DOTGOV** ($\mu = 500$)	
	MAP	P@10	MAP	P@10	MAP	P@10
Previous Work						
DirichletLM	0.2269	0.3960	0.2171	0.4080	0.1873	0.2460
NTLM-skipgram	**0.2427**	0.4100	0.2266	0.4240	0.1932	0.2500
NTLM-cbow	0.2418	**0.4193**	0.2262	0.4227	0.1916	0.2480
Our Results						
DirichletLM	0.1840	0.3060	**0.2731**	**0.4481**	**0.1943**	**0.2560**
NTLM-skipgram	0.1000	0.1712	0.1296	0.2480	0.0828	0.1200
NTLM-cbow	0.1081	0.1783	0.1320	0.2595	0.0876	0.1380
NTLM-Wordllama	0.0668	0.0713	0.0892	0.1275	0.0516	0.0610

DirichletLM Effectiveness: Our implementation of the DirichletLM shows mixed results compared to the previous work. On the AP88-89 dataset, our MAP score (18.40) is lower than the previous result (22.69), indicating a slight drop in effectiveness. However, on the WSJ87-92 dataset, our MAP score (27.31) is higher than the previous result (21.71), suggesting an improvement. On the DOTGOV dataset, the effectiveness is comparable, with our MAP score (19.43) slightly higher than the previous result (18.73).

NTLM-skipgram Effectiveness: Our NTLM-skipgram model has a lower effectiveness result compared to the previous work across all datasets. On the AP88-89 dataset, our MAP score (10.00) is statistically lower than the previous result (24.27). Similarly, on the WSJ87-92 dataset, our MAP score (12.86) is much lower than the previous result (22.66). The same trend is observed on the DOTGOV dataset, where our MAP score (11.32) is lower than the previous result (19.32).

NTLM-cbow Effectiveness: Similar to the NTLM-skipgram model, our NTLM-cbow model also underperforms compared to the previous work. On the AP88-89 dataset, our MAP score (10.18) is lower than the previous result (24.18). On the WSJ87-92 dataset, our MAP score (13.52) is lower than the previous result (22.62). On the DOTGOV dataset, our MAP score (13.16) is lower than the previous result (19.16).

The lower effectiveness of our NTLM method (skipgram and cbow) compared to the previous work could be attributed to subtle differences in the model architecture or implementation. For example, variations in how embeddings are initialized, updated, or normalized might lead to discrepancies in effectiveness. Additionally, the previous work may have used a different codebase which could introduce differences in optimization algorithms, numerical precision, or other implementation-specific details. These factors, even if minor, can impact the final results, explaining the observed effectiveness gap.

5.4 NTLM Evaluation Result

The evaluation results as shown in Table 5.4 demonstrates the effectiveness of the Dirichlet-smoothed language model and the NTLM using the following embedding Word2Vec, monoBERT, Roberta, GPT2, and WordLlama method across multiple datasets. The initial ranked Dirichlet model result shows better effectiveness in datasets like WSJ, DOTGOV, fiqa, msmarco-trec-dl-2020, scifact, nfcorpus, and webis-touche2020. For example, in webis-touche2020, Dirichlet model scores a MAP of 0.244, a P@10 of 0.406 and in scifact, it scores a MAP of 0.581 and nDCG@10 of 0.621.

Among the NTLM, monoBERT input embedding shows better effectiveness in some datasets, such as DOTGOV, where it achieves the highest P@10 (0.230) and nDCG @100 (0.314). But, it scores low effectiveness in fiqa, with a MAP of only 0.066, compared to Dirichlet's 0.167 and WordLlama's 0.160. The NTLM using the input embedding from transformer models like Roberta and GPT2 shows mixed results but still lower than the Dirichlet smoothed model and the NTLM when we use the monoBERT embedding. The NTLM using GPT2 input embedding was effective well in quora (MAP: 0.448), the NTLM using Roberta input embedding struggles in webis-touche2020, achieving the lowest MAP (0.070) and nDCG@10 (0.100).

Despite its strength in word translation, the Google New word2vec model was not as much as effective as the Dirichlet, the NTLM with monoBERT embedding, and Wordllama method. For instance, in fiqa, Word2Vec achieves a MAP of 0.096, much lower than the Dirichlet model and the WordLlama method. Compared with the other NTLM it

scores better effectiveness in webis-touche2020 MAP of 0.184. In contrast, the WordLlama method shows a good and competitive effectiveness result in the AP, quora, and trec-covid datasets. In quora, WordLlama method achieves a MAP of 0.705 and an nDCG@10 of 0.751, more effective than the Dirichlet model (MAP: 0.469, nDCG@10: 0.508). Similarly, in trec-covid, it achieves the highest P@10 (0.624) and nDCG@10 (0.588). Refer Table 5.4 for a detailed and complete analysis.

Table 5.4: Effectiveness Dirichlet smoothed and NTLM model across different datasets for Top 100 Documents. Bold values indicate the highest scores achieved for each metric across different models for a given dataset. Note that the word2vec used in this table is the Google News model

Dataset	Model	MAP	MAP@10	P@10	P@100	nDCG@10	nDCG@100
AP	Dirichlet	0.121	0.043	0.307	0.204	0.319	0.329
	AP-Skipgram	0.099	0.028	0.238	0.203	0.243	0.305
	AP-CBOW	0.100	0.029	0.246	0.203	0.250	0.306
	word2vec	0.087	0.021	0.195	0.203	0.199	0.291
	monoBERT	0.100	0.029	0.241	0.201	0.251	0.305
	Roberta	0.072	0.013	0.157	0.203	0.157	0.269
	GPT2	0.080	0.014	0.185	0.203	0.181	0.281
	wordllama	**0.127**	**0.048**	**0.316**	**0.205**	**0.340**	**0.340**
WSJ	Dirichlet	**0.202**	**0.072**	0.449	0.268	0.485	**0.434**
	WSJ-Skipgram	0.154	0.038	0.355	**0.269**	0.373	0.391
	WSJ-CBOW	0.154	0.038	0.356	**0.269**	0.373	0.391
	word2vec	0.143	0.028	0.317	**0.269**	0.321	0.375
	monoBERT	0.162	0.047	0.363	**0.269**	0.379	0.396
	Roberta	0.134	0.027	0.284	**0.269**	0.285	0.367
	GPT2	0.145	0.033	0.341	**0.269**	0.339	0.380
	wordllama	0.186	0.060	**0.464**	0.268	**0.489**	0.425
DOTGOV	Dirichlet	**0.133**	**0.086**	0.196	0.085	0.250	0.290
	DOTGOV-Skipgram	0.103	0.047	0.152	0.108	0.177	0.288
	DOTGOV-CBOW	0.105	0.049	0.156	0.108	0.182	0.290
	word2vec	0.091	0.045	0.174	0.094	0.201	0.266
	monoBERT	0.132	0.075	**0.230**	0.108	**0.255**	**0.314**
	Roberta	0.078	0.021	0.144	**0.109**	0.149	0.254
	GPT2	0.083	0.023	0.154	**0.109**	0.155	0.261
	wordllama	0.108	0.059	0.202	0.083	0.236	0.275
fiqa	Dirichlet	**0.167**	**0.156**	0.050	0.010	0.197	0.251
	word2vec	0.096	0.082	0.029	**0.011**	0.110	0.193
	monoBERT	0.066	0.051	0.023	**0.011**	0.074	0.164
	Roberta	0.058	0.044	0.020	**0.011**	0.063	0.154
	GPT2	0.104	0.090	0.035	**0.011**	0.125	0.202
	wordllama	0.160	0.152	**0.056**	0.010	**0.204**	**0.253**
quora	Dirichlet	0.469	0.447	0.113	**0.020**	0.508	0.574
	word2vec	0.348	0.320	0.098	**0.020**	0.394	0.486
	monoBERT	0.377	0.352	0.096	**0.020**	0.425	0.511
	Roberta	0.171	0.140	0.062	**0.020**	0.199	0.334
	GPT2	0.448	0.423	0.115	**0.020**	0.501	0.572
	wordllama	**0.705**	**0.689**	**0.156**	**0.020**	**0.751**	**0.769**
msmarco-trec-dl-2020	Dirichlet	**0.338**	**0.177**	**0.520**	0.190	**0.483**	**0.544**
	word2vec	0.201	0.071	0.322	**0.192**	0.249	0.421
	monoBERT	0.272	0.138	0.382	0.190	0.367	0.496

Continued on next page

Table 5.4 – *Continued from previous page*

Dataset	Model	MAP	MAP@10	P@10	P@100	nDCG@10	nDCG@100
	Roberta	0.223	0.090	0.364	0.190	0.314	0.444
	GPT2	0.248	0.101	0.404	0.190	0.359	0.471
	wordllama	0.289	0.146	0.487	0.182	0.454	0.511
scifact	Dirichlet	**0.581**	**0.574**	**0.082**	**0.010**	**0.621**	**0.650**
	word2vec	0.331	0.316	0.053	**0.010**	0.359	0.441
	monoBERT	0.400	0.387	0.063	**0.010**	0.436	0.500
	Roberta	0.288	0.270	0.047	**0.010**	0.312	0.407
	GPT2	0.343	0.325	0.055	**0.010**	0.371	0.455
	wordllama	0.498	0.488	0.076	**0.010**	0.541	0.585
trec-covid	Dirichlet	0.033	0.006	0.328	0.275	0.297	0.249
	word2vec	0.037	0.006	0.428	**0.309**	0.372	0.284
	monoBERT	0.039	0.007	0.460	**0.309**	0.411	**0.291**
	Roberta	0.036	0.005	0.342	**0.309**	0.289	0.271
	GPT2	0.037	0.006	0.342	**0.309**	0.303	0.274
	wordllama	**0.044**	**0.013**	**0.624**	0.265	**0.588**	0.290
nfcorpus	Dirichlet	**0.119**	**0.092**	**0.230**	**0.063**	**0.290**	**0.249**
	word2vec	0.061	0.037	0.088	**0.063**	0.104	0.169
	monoBERT	0.094	0.067	0.184	**0.063**	0.225	0.219
	Roberta	0.087	0.063	0.154	**0.063**	0.189	0.204
	GPT2	0.091	0.065	0.164	**0.063**	0.203	0.211
	wordllama	0.111	0.085	0.223	0.062	0.274	0.237
webis-touche2020-V2	Dirichlet	**0.244**	**0.185**	**0.406**	0.088	**0.451**	**0.485**
	word2vec	0.184	0.119	0.320	**0.089**	0.335	0.428
	monoBERT	0.132	0.074	0.222	**0.089**	0.242	0.374
	Roberta	0.070	0.022	0.102	**0.089**	0.100	0.292
	GPT2	0.150	0.083	0.241	**0.089**	0.252	0.389
	wordllama	0.181	0.113	0.273	0.087	0.308	0.418

5.5 Estimation of the Translation Probabilities

A key component of the translation language model is learning the word-to-word translation probability, $p_t(w|u)$, and it has a high impact on the overall effectiveness of the model. So, it is necessary to assess the effectiveness of all models. To achieve this, we use different types of input embeddings from transformer models and word embeddings to estimate translation probabilities. We compute the cosine similarity between the vector representations of the target word w and the candidate word u. These similarities are then normalized to create a probability distribution over all possible translations. Understanding the impact of word-to-word translation using different embeddings is very important, as it forms the core of the translation language model equation. This process captures the semantic relationship between query and document words, potentially impacting the effectiveness of the NTLM.

In the analysis, we estimated the translation probabilities for different words across different embedding models. As shown in Table 5.5, nearly all models demonstrated good word translation for very common words, assigning a high probability to self-translation $p_t(w|w)$, which is generally ranked higher among possible translations. A comparative review of the GPT-2 and CBOW models reveals that they struggle to assign the self-translation target word to the higher rank. In contrast, the Skip-Gram, BERT, Mono

Table 5.5: Comparison of word translations, along with translation probabilities, for Common words across different Models

skipgram (Google News 2013)				cbow (Oil and Gas Corpus)				BERT base			
w_1 = forest		w_2 = visitor		w_1 = forest		w_2 = visitor		w_1 = forest		w_2 = visitor	
u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$
forests	0.5017	visitors	0.3457	woodland	0.1876	tourist	0.2674	forest	0.1801	visitor	0.1677
forested	0.1045	tourist	0.1443	rainforest	0.1653	people	0.1479	forests	0.1125	visitors	0.1195
forestland	0.0820	Visitor	0.1345	deciduous_forest	0.1503	landowner	0.1286	woodland	0.0949	visitation	0.0912
forestry	0.0734	vistor	0.1053	coniferous_forest	0.1250	volunteer	0.1001	forestry	0.0896	tourist	0.0900
Forests	0.0696	tourists	0.0785	vegetation	0.1034	foreign_investment	0.0971	woods	0.0896	traveller	0.0887
woodlands	0.0650	traveler	0.0751	montane_forest	0.0962	person	0.0919	rainforest	0.0889	intruder	0.0886
rainforest	0.0517	vistors	0.0701	grassland	0.0913	hike	0.0857	jungle	0.0879	guests	0.0879
Mono BERT				RoBERTa				GPT-2			
forest	0.2024	visitor	0.1814	forest	0.2849	Visit	0.1163	forest	0.2133	Vis	0.1122
forests	0.1118	visitors	0.1238	forestation	0.2849	Guest	0.1142	forestation	0.2133	Visit	0.1087
woods	0.0918	visiting	0.0904	Forest	0.1625	resident	0.1096	Forest	0.1404	vis	0.0855
woodland	0.0882	visitation	0.0890	woods	0.1383	Vis	0.1052	woods	0.1243	quickShip	0.0778
forestry	0.0865	intruder	0.0879	pine	0.0981	Welcome	0.1004	Tree	0.0938	ÿ	0.0759
rainforest	0.0856	visits	0.0875	leaf	0.0914	Customer	0.0967	wild	0.0903	ý	0.0759
jungle	0.0855	tourist	0.0874	Park	0.0914	Tour	0.0855	Wild	0.0868	û	0.0759

Table 5.6: Comparison of translation probabilities for Not-Common Words across different Models

Skipgram (Google News 2013)				CBOW (Oil and Gas Corpus)				BERT Base			
w_1 = radioactive		w_2 = pneumonia		w_1 = radioactive		w_2 = pneumonia		w_1 = radioactive		w_2 = pneumonia	
u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$	u	$p(w\|u)$
radioactive_material	0.2135	respiratory_infection	0.1915	radioactive_decay	0.2230	unavailable	0.0	radioactive	0.7443	pneumonia	0.7312
radioactivity	0.1940	bacterial_pneumonia	0.1712	uranium_thorium	0.1496	unavailable	0.0	contaminated	0.0316	tuberculosis	0.0479
radioactive_materials	0.1535	viral_pneumonia	0.1360	radioactivity	0.1293	unavailable	0.0	unavailable	0.0	influenza	0.0373
radiation	0.1209	lung_infection	0.1289	heat-producing_element	0.1073	unavailable	0.0	unavailable	0.0	cholera	0.0308
radioactive_substances	0.0959	respiratory_illness	0.1098	heat-producing	0.1023	unavailable	0.0	unavailable	0.0	infections	0.0287
radioactive_wastes	0.0753	bacterial_infection	0.0984	radioactive_nuclide	0.1005	unavailable	0.0	unavailable	0.0	malaria	0.0280
radioactive_isotopes	0.0738	bronchitis	0.0870	radiogenic	0.0944	unavailable	0.0	unavailable	0.0	unavailable	0.0
Mono BERT				RoBERTa				GPT-2			
radioactive	0.1770	pneumonia	0.1739	nuclear	0.1333	cancer	0.1170	nuclear	0.1117	Ę	0.0751
contaminated	0.0948	tuberculosis	0.0974	atomic	0.1166	pox	0.0817	Ē	0.0774	Ē	0.0751
poisonous	0.0918	influenza	0.0936	electric	0.0996	itis	0.0762	ę	0.0774	ċ	0.0751
unavailable	0.0	infections	0.0933	obyl	0.0742	rosis	0.0737	Ą	0.0774	ă	0.0751
unavailable	0.0	cholera	0.0925	agnetic	0.0692	philis	0.0735	ö	0.0750	Ġ	0.0751
unavailable	0.0	asthma	0.0914	utonium	0.0670	monary	0.0721	ú	0.0750	Ă	0.0751
unavailable	0.0	leukemia	0.0904	uclear	0.0665	thritis	0.0706	ü	0.0750	ę	0.0751

BERT, and RoBERTa models perform well, achieving higher self-translation probabilities and better rankings.

The difference in model effectiveness emerges when evaluating less common words, as shown in Table 5.6. The analysis of various terms such as "hypertension," "pesticide," and "geology" shows variations in translation effectiveness. The Skip-Gram model (trained on Google News 2013) is consistently more effective than the other models, likely due to its training on a substantial dataset of approximately 100 billion words, resulting in a model size of around 3.3 GB. While this model yields excellent results, it requires considerably more processing time compared to the other embeddings, indicating a trade-off between efficiency and computational speed.

Following the Skip-Gram model, both MonoBERT and RoBERTa exhibited better effectiveness. Particularly, they attempted to split unfamiliar words into recognizable subwords within their vocabularies. For example, for the word "hypertension," the models generated "hyper" as a potential translation candidate. In contrast, despite GPT-2's vocabulary of approximately 50,000 vocabulary size, it achieved the lowest effectiveness among the models evaluated. This suggests that model architecture and the size of the training data could impact translation effectiveness, especially for less common words.

Based on the parameters detailed in Table 4.2 in the methodology chapter, the Word2Vec

models were trained using two distinct datasets, AP88-89 and WSJ87-92, to examine how domain-specific training influences the effectiveness of word embeddings. The training parameters, including embedding dimensions, window size, and number of epochs, were kept consistent across both datasets to ensure a fair comparison. The AP88-89 dataset, with a size of 0.72 GB and 242,918 documents, is larger than the WSJ87-92 dataset, which has a size of 0.52 GB and 173,252 documents, indicating greater linguistic diversity and coverage.

Table 5.7: Static Word Embedding Models Result Trained on AP and WSJ Datasets

Corpus	Model	Voc.Size	Dimensions	Window Size	Size(MB)
WSJ	Skip-gram	81,970	300	5	95
WSJ	CBOW	81,486	300	5	95
AP	Skip-gram	127,285	300	5	146
AP	CBOW	126,764	300	5	146

These distinctions are reflected in the effectiveness of the embeddings, particularly in word translation tasks. The Word2Vec embeddings trained on AP88-89 demonstrate greater effectiveness than those trained on WSJ87-92, particularly for rare words. This improvement is largely due to AP88-89 being a larger dataset, offering greater vocabulary diversity, which enables the model to learn richer and more generalizable word representations. A comparison between Word2Vec embeddings and transformer-based input embeddings revealed comparable effectiveness in word translation tasks. This suggests that while transformer embeddings benefit from advanced architectures, static embeddings like Word2Vec can achieve similar effectiveness when trained on large and diverse datasets such as AP88-89.

Table 5.8: Comparison of Translation Probabilities for Common and Rare Words in WSJ and AP Datasets using Skipgram and CBOW Models.

AP Skipgram						**AP CBOW**					
w_1 = forest		**w_2 = visitor**		**w_3 = pesticide**		**w_1 = forest**		**w_2 = visitor**		**w_3 = pesticide**	
u	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$
forests	0.3810	visitors	0.4131	pesticides	0.4982	forests	0.4168	visitors	0.5099	pesticides	0.4087
bakersnoqualmie	0.2049	tourist	0.1325	phosdrin	0.1089	park	0.1937	tourists	0.1041	aldicarb	0.1220
timber	0.1046	visits	0.0909	malathion	0.0724	wildlife	0.0962	traveler	0.1009	insecticide	0.1074
wilderness	0.0647	tourists	0.0879	residues	0.0708	lodgepole	0.0679	tourist	0.0699	captan	0.0797
acres	0.0627	visiting	0.0752	aldicarb	0.0658	lightningsparked	0.0606	guests	0.0580	fungicide	0.0738
bridgerteton	0.0617	visit	0.0727	chemicals	0.0657	timber	0.0600	resident	0.0559	fungicides	0.0724
forestry	0.0607	visited	0.0642	cancercausing	0.0599	wilderness	0.0577	guest	0.0517	ebdc	0.0694
WSJ Skipgram						**WSJ CBOW**					
w_1 = forest		**w_2 = visitors**		**w_3 = pesticide**		**w_1 = forest**		**w_2 = visitors**		**w_3 = pesticide**	
u	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$
forestproducts	0.2043	tourists	0.3714	pesticides	0.2513	sonoco	0.1887	visitors	0.3478	pesticides	0.3409
forests	0.1418	tourist	0.1681	residues	0.2060	forests	0.1640	guests	0.1214	cancer-causing	0.1480
timber	0.1385	visitor	0.1093	isocyanate	0.1208	timber	0.1533	fourthfloor	0.1059	residues	0.1455
portlandbased	0.1312	guests	0.0909	cancercausing	0.1137	lawncare	0.1103	tourists	0.0985	herbicide	0.0873
northwood	0.1101	blitar	0.0843	cyanazine	0.0805	forestproducts	0.1099	balcony	0.0925	cyhexatin	0.0779
timberlands	0.0956	sightseers	0.0619	ebdc	0.0792	fishery	0.1002	guest	0.0854	formaldehyde	0.0680
diboll	0.0913	shooed	0.0579	cyhexatin	0.0757	logging	0.0927	tour	0.0764	herbicides	0.0664
DOTGOV Skipgram						**DOTGOV CBOW**					
w_1 = forest		**w_2 = visitor**		**w_3 = pesticide**		**w_1 = forest**		**w_2 = visitor**		**w_3 = pesticide**	
u	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$
forests	0.3420	visitors	0.4541	pesticides	0.4792	forests	0.4178	visitors	0.5109	pesticides	0.4097
timber	0.2259	visits	0.1335	phosdrin	0.1099	park	0.1947	tourists	0.1051	aldicarb	0.1230
timberland	0.1056	tourist	0.0919	insecticide	0.0734	wildlife	0.0972	traveler	0.1019	insecticide	0.1084
forestry	0.0657	tourists	0.0889	residues	0.0718	lodgepole	0.0689	tourist	0.0709	captan	0.0807
acres	0.0637	visiting	0.0762	aldicarb	0.0668	acres	0.0616	guests	0.0590	fungicide	0.0748
wilderness	0.0627	visited	0.0737	chemicals	0.0667	timber	0.0610	resident	0.0569	fungicides	0.0734
northwood	0.0617	visit	0.0652	cancercausing	0.0609	wilderness	0.0587	tour	0.0527	ebdc	0.0704

Table 5.9: Word-to-Word Translation Probabilities for Common and Rare Words using Wordllama Input Embedding Model.

Wordllama l3_supercat.safetensors model												
w_1 = forest		**w_2 = visitor**		**w_3 = car**		**w_4 = radioactive**		**w_5 = pesticide**		**w_6 = hypertension**	
u	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$	**u**	$p(w\|u)$
Forest	0.6717	Visitor	0.1268	Car	0.1268	no translation	—	no translation	—	no translation	—
forest	0.6559	visitor	0.1246	cars	0.1259	—	—	—	—	—	—
Forest	0.6527	visitors	0.1230	Car	0.1250	—	—	—	—	—	—
forests	0.5458	Visitors	0.1222	car	0.1240	—	—	—	—	—	—
forestation	0.5363	_visitor	0.1203	(car	0.1239	—	—	—	—	—	—
trees	0.4237	visiting	0.1137	-car	0.1226	—	—	—	—	—	—
fore	0.4188	—	—	/car	0.1223	—	—	—	—	—	—

Table 5.9, shows that the WordLlama input embedding `l3_supercat_1024_.safetensors` assigns high self-translation probabilities for common words but lacks diversity in generating semantically related alternatives. It also struggles with rare words like "radioactive" and "pesticide," failing to produce valid translations. This limitation suggests weaker handling of both low-frequency terms and broader lexical variation, unlike models such as GPT2 and RoBERTa, which leverage subword decomposition. While effective for direct self-translation, WordLlama may require improvements to enhance word diversity and rare-word representations.

Chapter 6

Discussion

6.1 RQ1: Comparative Analysis of the NTLM

This research question investigates the effectiveness of integrating Word2Vec and transformer based input embeddings into the NTLM for improving document reranking. The question investigates whether incorporating word embedding into the translation language model can better capture semantic relationships between queries and documents, potentially offering improved semantic matching and addressing vocabulary mismatch issues.

The experimental results provide valuable insights into the effectiveness of the reranking result of the NTLM compared to the initial ranker, the Dirichlet-smoothed language model. Table 5.4 shows the retrieval results for the Dirichlet-smoothed language model and the NTLM using Word2Vec, input embeddings, and the WordLlama method, with the best overall results highlighted in bold. Despite its simplicity, the Dirichlet model consistently demonstrates strong effectiveness across multiple datasets. This shows the effectiveness of traditional probabilistic methods in document ranking tasks, particularly in scenarios where exact term matching is important. However, the Dirichlet model does not achieve the best results for the AP, Quora, and TREC-COVID datasets, suggesting limitations in handling some specific datasets.

The effectiveness of reranking results of the NTLM varies across different datasets. Overall, the NTLM with monoBERT input embedding shows better effectiveness compared to the other neural models like the NTLM with word2vec, GPT2, and RoBERTa embedding. For example, in the DOTGOV dataset, the NTLM with monoBERT input embedding achieves even better results than the other NTLM and the WordLlama method. This could be the NTLM with monoBERT is a fine-tuned variant of the BERT model, which might enable the model to increase its effectiveness. Additionally, the monoBERT input embedding has good translation as shown in the word translation Table 5.6. The Roberta input embedding shows less effectiveness in translating the word into possible candidate words that have semantically related translated words. As a result, the effectiveness of the reranking NTLM using the Roberta model is the lowest. Even though the NTLM using GPT2 input embedding has lower effectiveness compared to the other models, but still has better effectiveness than the NTLM using the Roberta input embedding.

The NTLM relatively lower effectiveness compared to the Dirichlet smoothed model might be due to these factors. First, as demonstrated in prior work by Karimzadehgan and Zhai [17], translation language models based on mutual information have been shown to be more effective than the simple language models like the Dirichlet model, which rely on exact term matching. While the NTLM approaches treat query generation as a translation process and leverage word embedding to improve the estimation of translation probabilities, the effectiveness of these methods depends heavily on the quality of the translation probability estimation. In our case, the estimation of $P(u|w)$ using cosine similarity may not sufficiently improve the overall effectiveness of the NTLM model. From the analysis, we observed that models that have lower effectiveness in word-to-word translation have lower effectiveness in the NTLM. So this weakness might impact the overall NTLM effectiveness. Second, the initial retrieval step using PyTerrier may not provide an optimal foundation for the NTLM reranking. If the initial retrieval results are suboptimal, the reranking process is inherently limited in its ability to improve reranking effectiveness.

6.2 RQ2: Impact of Embedding Variations and Fine-Tuned Transformers

In this research question, we investigate how different Word2Vec configurations, input embeddings from various transformer models, and fine-tuned input embedding from transformers like MonoBERT affect retrieval effectiveness. Specifically, we analyze whether embedding characteristics, architectural choices, and training configurations enhance semantic similarity capture and reranking effectiveness.

As shown in Table 5.4, we tested multiple Word2Vec embeddings and transformer-based input embeddings. Overall, the differences in retrieval effectiveness were minor, but some patterns were shown. The experiment compared the reranking effectiveness of the NTLM using Word2Vec embeddings with the NTLM using transformer-based input embeddings (e.g., MonoBERT, GPT-2, RoBERTa) to assess the impact of different architectures and training methods.

We trained Word2Vec embeddings on three datasets (AP, WSJ, and DOTGOV) using consistent parameters: a 300-dimensional embedding space, a window size of 5, a minimum word frequency of 2, 15 negative samples, and 50 epochs (30 for DOTGOV due to its larger size), as detailed in Table 4.1. The results showed little variation in effectiveness across these datasets. However, the CBOW model was slightly more effective than the Skip-gram model, though both remained less effective than the initial ranked Dirichlet-smoothed model. Furthermore, Word2Vec embeddings trained on domain-specific datasets (AP, WSJ, and DOTGOV) were slightly more effective than the general pre-trained Google News Word2Vec model. This suggests that domain-specific training leads to improved effectiveness, even if the differences are small.

Unlike Word2Vec, transformer-based embeddings offer higher-dimensional representations and richer semantic relationships. We tested MonoBERT-large-msmarco (1024 dimensions), GPT-2 (768 dimensions), and RoBERTa-base (768 dimensions). These models

benefit from large-scale pretraining on diverse corpora, enhancing their ability to capture semantic meaning. Among the transformer input embeddings, the NTLM using the MonoBERT embedding was the most effective, particularly on the DOTGOV dataset. As shown in Table 4.3, despite having a smaller vocabulary size (30,000 words) compared to GPT-2 and RoBERTa (50,000 words), the NTLM using the MonoBERT that was fine-tuned on the MS MARCO datasets showed higher retrieval effectiveness. This highlights the role of fine-tuning in improving ranking effectiveness, potentially more than vocabulary size. Additionally, higher-dimensional embeddings, such as MonoBERT's 1024 dimensions, provided richer semantic representations. However, the NTLM using the monoBERT embedding does not do well for msmarco-trec-dl-2020 datasets compared to the Dirichlet smoothed model. The NTLM using the RoBERTa input embedding, despite its larger vocabulary, was the least effective among the other NTLM using the transformer-based embeddings. This suggests that a larger vocabulary alone does not necessarily lead to better retrieval effectiveness.

6.3 RQ3: Effectiveness of WordLlama for Reranking

In this research question, we investigate how effectively the WordLlama method and WordLlama input embeddings, designed for lightweight and efficient word representations, enhance the effectiveness of the NTLM reranking tasks. Specifically, we evaluate whether WordLlama's compact input embeddings, extracted from LLaMA 3, improve semantic representation and retrieval effectiveness compared to the Dirichlet-smoothed model, Word2Vec, and transformer-based input embeddings.

As discussed in Section 4.7, we hypothesized that WordLlama's input embeddings could enhance the NTLM effectiveness. This assumption was based on the idea that WordLlama's lightweight yet high-quality embeddings, derived from LLaMA token representations, could serve as effective word-level features for the NTLM probabilistic calculations. To test this, we used the best-effective input embedding, `l3_supercat_1024.safetensors`. However, our experiments showed that WordLlama's word-to-word translation, particularly for less common words, was suboptimal, meaning it struggles to generate semantically richer candidate words for a given target word. This limitation carried over to the NTLM approach, where WordLlama's input embeddings resulted in the lowest effectiveness among all models, as shown in Table 5.3.

In addition to evaluating the embeddings, we also analyzed the effectiveness of the WordLlama method, as proposed by Miller [23], for ranking documents based on query-document similarity. This framework, which reuses components from large language models (LLMs) to create compact and effective word representations, showed mixed results. Overall, the WordLlama method was effective and better than all the NTLM in reranking across most datasets. Especially, its effectiveness was particularly strong for short queries and documents, as shown in datasets like Quora Table 5.4. This is likely due to Quora's shorter average query and document lengths as shown in Table 3.1. However, the WordLlama method struggled with longer queries and documents, showing lower effectiveness than the initial Dirichlet language model, except for the AP, TREC-COVID, and Quora datasets. These results suggest that while WordLlama does well in sentence-level similarity and

ranking tasks, it is less effective when dealing with longer query-document datasets.

These findings highlight both the strengths and limitations of the WordLlama method. While its lightweight design and efficient embeddings make it an appealing alternative to larger models like Word2Vec, its effectiveness is highly dependent on dataset characteristics and task requirements.

Chapter 7

Conclusion

This thesis has shown the effectiveness of different models for document ranking with a focus on the Dirichlet-smoothed language model, the NTLM with various embeddings, and the WordLlama framework. The research provides key insights into the strengths and limitations of these approaches, evaluating their impact on retrieval and reranking effectiveness across multiple datasets. The experimental results showed the importance of embedding types and configurations, highlighting their influence on the overall effectiveness of document ranking. This chapter summarizes the key findings of the thesis.

The integration of Word2Vec and transformer-based input embeddings into the NTLM was assessed for their effectiveness in document reranking. The results showed that while the NTLM benefits from word embedding for semantic matching and resolving vocabulary mismatches, it often produces lower effectiveness in reranking compared to the initial ranking Dirichlet-smoothed language model. The Dirichlet model maintained effectiveness across multiple datasets. In contrast, the NTLM effectiveness varied depending on the dataset and the choice of embeddings. Among the NTLM, MonoBERT input embedding achieved the best reranking effectiveness, while the RoBERTa-based NTLM was the least effective due to its weaker word translation ability. These findings suggest that the NTLM effectiveness is influenced by the effectiveness of word translation estimation and the alignment of embeddings with the specific task or domain.

The experiment comparing WordLlama's reranking effectiveness showed that, despite its compact embeddings, WordLlama's input embedding was the least effective among all NTLM. Its weaker word-to-word translation ability limited its effectiveness in generating semantically rich candidates. However, WordLlama's ranking method was effective across most datasets, especially in short-query datasets like Quora. This suggests that WordLlama is effective for sentence-level ranking tasks.

In the word translation analysis, we found that most models demonstrated high effectiveness in translating common words, assigning strong probabilities to self-translation $p_t(w|w)$ and ranking them highly among possible candidates. The Skip-Gram, CBOW, BERT, MonoBERT, and RoBERTa models were effective, achieving higher self-translation probabilities. However, effectiveness varied for less common words. Google News showed higher effectiveness than others due to its rich vocabulary, handling even rare or uncommon words. MonoBERT and RoBERTa also performed effectively, using subword tokenization to process unfamiliar words. In contrast, GPT-2, despite its large vocab-

ulary, was the least effective for uncommon words. This highlights the importance of model architecture and training data size in word translation tasks.

From this research, we learned that the success of a ranking model isn't just about having a large vocabulary or complex embeddings. The most important factors are how well the embeddings fit the specific task and how well the model can handle both translating common and rare words. It also showed that fine-tuning and domain-specific training have an impact on improving ranking effectiveness.

In conclusion, this thesis demonstrates the integration of transformer-based input embeddings into the NTLM and their impact on retrieval effectiveness. While the Dirichlet-smoothed language model remains a strong initial ranker, the NTLM method requires further refinement to enhance semantic matching and improve its effectiveness. Fine-tuned transformer embeddings, particularly MonoBERT, improve reranking but show varying effectiveness across datasets. Although we expected the NTLM to achieve better effectiveness than the Dirichlet model in reranking by leveraging embeddings, the Dirichlet model consistently demonstrated higher effectiveness. This suggests that the word embeddings still have room for improvement.

This thesis has identified several areas for future research to address the limitations of current models and further advance the field of document ranking and information retrieval. One promising direction is the integration of contextualized into the NTLM framework. Contextualized embeddings, such as those from transformer-based models like BERT, RoBERTa, or GPT, can capture semantic and contextual relationships compared to static embeddings like Word2Vec and input embedding from a transformer model. For example, contextualized embeddings could replace the word2vec and input embedding to estimate the translation probability, enabling the model to better capture the query-document interactions.

Another promising direction for future work is the extension of the NTLM framework to incorporate neural query expansion [7] techniques. This could involve developing methods to automatically identify and add relevant terms from the document collection to the original query, thereby improving retrieval performance. For instance, leveraging contextualized embeddings or transformer-based models to generate query expansions that capture semantic and contextual relationships could help address vocabulary mismatch and improve recall. Such advancements would enhance the NTLM ability to handle complex queries and improve its overall retrieval effectiveness.

Bibliography

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[2] Adam Berger. Information retrieval as statistical translation. 1999.

[3] Michael W Berry, Zlatko Drmac, and Elizabeth R Jessup. Matrices, vector spaces, and information retrieval. *SIAM review*, 41(2):335–362, 1999.

[4] Alexander Bondarenko, Lukas Gienapp, Maik Fröbe, Meriem Beloucif, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, et al. Overview of touché 2021: argument retrieval. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction: 12th International Conference of the CLEF Association, CLEF 2021, Virtual Event, September 21–24, 2021, Proceedings 12*, pages 450–467. Springer, 2021.

[5] Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. A full-text learning to rank dataset for medical information retrieval. In *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*, pages 716–722. Springer, 2016.

[6] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[7] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *Acm Computing Surveys (CSUR)*, 44(1):1–50, 2012.

[8] Kenneth Church and Robert L Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational linguistics*, 19(1):1–24, 1993.

[9] Nick Craswell. Mean reciprocal rank. *Encyclopedia of database systems*, pages 1703–1703, 2009.

[10] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the trec 2020 deep learning track, 2021.

[11] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Norbert Fuhr. Probabilistic models in information retrieval. *The computer journal*, 35(3):243–255, 1992.

[13] Lukas Galke, Ahmed Saleh, and Ansgar Scherp. Word embeddings for practical information retrieval. In *Informatik 2017*, pages 2155–2167. Gesellschaft für Informatik, 2017.

[14] Christophe Van Gysel, Maarten De Rijke, and Evangelos Kanoulas. Neural vector spaces for unsupervised information retrieval. *ACM Transactions on Information Systems (TOIS)*, 36(4):1–25, 2018.

[15] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *ACM SIGIR Forum*, volume 51, pages 243–250. ACM New York, NY, USA, 2017.

[16] Ehsan Kamalloo, Nandan Thakur, Carlos Lassance, Xueguang Ma, Jheng-Hong Yang, and Jimmy Lin. Resources for brewing beir: Reproducible reference models and statistical analyses. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 1431–1440, New York, NY, USA, 2024. Association for Computing Machinery.

[17] Maryam Karimzadehgan and ChengXiang Zhai. Estimation of statistical translation models based on mutual information for ad hoc information retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 323–330, 2010.

[18] Tom Kenter, Alexey Borisov, Christophe Van Gysel, Mostafa Dehghani, Maarten de Rijke, and Bhaskar Mitra. Neural networks for information retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1403–1406, 2017.

[19] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[20] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. Simplified data wrangling with ir datasets. In *SIGIR*, 2021.

[21] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. Www'18 open challenge: financial opinion mining and question answering. In *Companion proceedings of the the web conference 2018*, pages 1941–1942, 2018.

[22] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[23] D. Lee Miller. Wordllama: Recycled token embeddings from large language models, 2024.

[24] Bhaskar Mitra, Nick Craswell, et al. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, 2018.

[25] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th international conference on world wide web*, pages 1291–1299, 2017.

[26] University of Mannheim. Evaluation in information retrieval, 2020.

[27] Pinecone. Evaluation measures in information retrieval, 2012.

[28] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *ACM SIGIR Forum*, volume 51, pages 202–208. ACM New York, NY, USA, 2017.

[29] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[30] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[31] Stephen E Robertson, Evangelos Kanoulas, and Emine Yilmaz. Extending average precision to graded relevance judgments. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 603–610, 2010.

[32] Stephen E Robertson and Steve Walker. A probabilistic model of information retrieval. *Information Processing Management*, 36:809–840, 2000.

[33] G Salton. Introduction to modern information retrieval, 1983.

[34] Gerard Salton. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169, 1989.

[35] Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. Natural language understanding with the quora question pairs dataset. *arXiv preprint arXiv:1907.01041*, 2019.

[36] Mark D Smucker and James Allan. An investigation of dirichlet prior smoothing's performance advantage. Technical report, Citeseer, 2005.

[37] Mark D Smucker, David Kulp, and James Allan. Dirichlet mixtures for query estimation in information retrieval. *Center for Intelligent Information Retrieval*, 2005.

[38] Christophe Van Gysel and Maarten de Rijke. Neural vector spaces for unsupervised information retrieval. *arXiv preprint arXiv:1708.02702*, 2017.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, and Aidan N Gomez. L. u. kaiser, and i. polosukhin,"attention is all you need,". *Advances in neural information processing systems*, 30:5998–6008, 2017.

[40] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. Trec-covid: constructing a pandemic information retrieval test collection. In *ACM SIGIR Forum*, volume 54, pages 1–12. ACM New York, NY, USA, 2021.

[41] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. *arXiv preprint arXiv:2004.14974*, 2020.

[42] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[43] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM Sigir Forum*, volume 51, pages 268–276. ACM New York, NY, USA, 2017.

[44] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2(30):6, 2004.

[45] Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In *Proceedings of the 20th Australasian document computing symposium*, pages 1–8, 2015.

[46] Guido Zuccon, João R. M. Palotti, and Allan Hanbury. Query variations and their effect on comparing information retrieval systems. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24–28, 2016*, pages 691–700. ACM, 2016.