The present work was submitted to the
Visual Computing Institute (RWTH Aachen University)
and to the
Jülich Supercomputing Centre (Forschungszentrum Jülich)

Rheinisch-Westfälische Technische Hochschule Aachen
Informatik 13, Visual Computing Institute

# Exploring Linguistic Proximity in C4 Multilingual Data through Efficient Embedding Model Analysis and Visualization on HPC

Sahand Rahmdel
(Matr.-Nr.: 424069)

March 24, 2025

| | |
|---|---|
| 1st Examiner | Prof. Dr. Bastian Leibe |
| 2nd Examiner | Prof. Dr. Stefan Kesselheim |
| 1st Supervisor | Dr. Jiangtao Wang |
| 2nd Supervisor | Dr. Oleg Filatov |

**Abstract**

This thesis investigates the proximity of different languages and different language families by analysing how multilingual text data are represented in a shared latent space, focusing on the Colossal Clean Crawled Corpus (C4) with a multilingual extension (mC4). The main focus is to determine whether embeddings of different languages group together based on their linguistic families, topical content, or both. This is achieved through a high-performance computing (HPC) system to embed 6.1TB of textual data from 24 diverse languages. The BAAI bge-m3 embedding model served to create embeddings of dimension 1,024, which were stored in a vector database using ChromaDB to facilitate scalable analysis and querying.

Subsequent dimensionality reduction with t-distributed Stochastic Neighbor Embedding (t-SNE) allowed for the visualization of language clusters in two-dimensional space for a simpler and better understanding. Results reveal that similar thematic or topical content often drives the embedding model to generate vectors that lie close together, even from different languages. However, certain clusters reflect linguistic closeness—especially among languages from the same family—indicating that the model also recognizes linguistic features. Overall, the thesis uses multilingual embeddings to check the existence of any relation between the semantic representation of texts as vectors (embeddings) and the linguistic structure of the origin languages, demonstrating how HPC resources, combined with advanced embedding models, can efficiently handle large datasets and offer deeper insights into language proximity and topic similarity analysis.

# Acknowledgments

I would like to express my sincere gratitude to Prof. Dr. Bastian Leibe, Chair of Computer Vision at RWTH Aachen University, for his invaluable support and for providing me with the opportunity to conduct this research. His pioneering work in computer vision, object detection, and recognition has been a great inspiration, and I am deeply appreciative of his guidance throughout this project.

I am especially grateful to Prof. Dr. Stefan Kesselheim, head of the Simulation and Data Lab Applied Machine Learning and Artificial Intelligence at the Jülich Supercomputing Centre (JSC), for accepting me into his team, and providing me with this chance and opportunity. Working in such a profound environment has greatly enriched my research experience, and taught me a lot. I cannot express my gratitude enough for his support and help.

My deepest appreciation goes to my supervisor, Dr. Jiangtao Wang, for his unwavering support and patience in addressing my numerous questions and needs. His guidance has been instrumental in the successful completion of this thesis.

I would also like to thank Dr. Oleg Filatov and Jan Ebert from the JSC team for their insightful ideas and guidance, which have significantly contributed to my work.

My heartfelt gratitude extends to my loving mother and sister, for their unwavering support and encouragement. I dedicate this work to the memory of my late father, whom I sadly lost to COVID-19.

Finally, I acknowledge the High-Performance Computing resources available at Forschungszentrum Jülich, which were essential for the successful execution of this project. I am profoundly grateful for the kindness and support of all the professors, colleagues, and friends who have accompanied me on this journey.

# Contents

# Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| AllenAI | Allen Institute for Artificial Intelligence |
| BAAI | Beijing Academy of Artificial Intelligence |
| BERT | Bidirectional Encoder Representations from Transformers |
| C4 | Colossal Clean Crawled Corpus |
| CPU | Central Processing Unit |
| GPU | Graphics Processing Unit |
| HPC | High-Performance Computing |
| IR | Information Retrieval |
| JSON | JavaScript Object Notation |
| LLM | Large Language Model |
| M2D2 | Massively Multi-Domain Dataset |
| mC4 | Multilingual Colossal Clean Crawled Corpus |
| NER | Named Entity Recognition |
| NLP | Natural Language Processing |
| NMT | Neural Machine Translation |
| OOM | Out-of-Memory |
| POS | Part-of-Speech |
| RNN | Recurrent Neural Network |
| RoBERTa | A Robustly Optimized BERT Pretraining Approach |
| SBERT | Sentence-BERT |
| t-SNE | t-distributed Stochastic Neighbor Embedding |
| UUID | Universally Unique Identifier |

# 1 Introduction

## 1.1 Motivation

There are vast amounts of data from many different languages across the internet, which can be used to train multilingual language models. However, to be able to use this data, it needs to be collected, cleaned, and processed [1]. The Colossal Clean Crawled Corpus - Common Crawl (C4) dataset is a large-scale dataset that contains roughly 750 GB of textual data from web pages in English [2]. Even though having the data is a good start, it is essential to gain knowledge about the data to know how it can be used effectively. The more is known about the dataset, the topics it covers, and how its topics are shared across different languages, the better it can be utilized in machine learning tasks [3].

One of the challenges of a large dataset like C4 is that the resources are not equally distributed for all of the languages [2]. While high-resource languages like English are well represented, some languages have significantly fewer resources [2]. This imbalance affects the performance of multilingual language models, particularly for low-resource languages [4].

Embedding models are a popular choice for representing textual data in a continuous vector space. These models are used in various natural language processing tasks, such as text classification, named entity recognition, and machine translation [5–7]. In embeddings, latent space refers to a mathematical space where data points (in this case, text representations) are embedded as vectors, capturing meaningful patterns in a compressed and abstract form. This representation is "latent" because it is not directly observable, but derived through machine learning models like embedding models [5].

This thesis leverages embedding models and t-SNE dimensionality reduction technique to visualize the relationships between languages in the C4 dataset. By using t-SNE to reduce the dimensionality of multilingual embeddings, this study seeks to identify whether languages with similar linguistic properties cluster together. Visualizing these clusters could help identify relationships in multilingual embeddings that are otherwise hidden. The gained insights contribute to understanding the effectiveness of multilingual embeddings in capturing linguistic proximity and may pro-

vide information about the content of the C4 dataset and topic overlaps between various languages for future research to improve the performance for low-resource languages.

## 1.2 Thesis Goal

This thesis aims to analyze the latent representations of multilingual datasets, specifically focusing on embeddings generated from the C4 multilingual dataset. By leveraging techniques such as t-Distributed Stochastic Neighbor Embedding (t-SNE) for visualization, this research provides insights into how languages cluster within shared latent spaces. This thesis investigates the types of clusters that emerge in the dataset and whether the clustering is by linguistic similarities and language families (shared language families) or by thematic and topical similarities, where data points are grouped based on similar subjects (e.g., Science, Health, etc.).

High-Performance Computing (HPC) systems are powerful computers that use parallel processing and specialized hardware to perform complex computations at high speeds. The practical aspect of this thesis involves efficient use of computational resources, employing parallel processing on an HPC system, and utilizing a vector database for scalable storage and analysis of embeddings. These contributions not only offer methodological advancements but also provide a foundation for future strategies in multilingual model training and evaluation.

## 1.3 Thesis Contributions

### *1.3.1 Multilingual Embedding and Distribution Process*

This thesis uses 24 selected languages from the mC4 dataset, from diverse language families and different resource availability. Text samples are embedded using the bge-m3 model, which is selected for its multilingual capabilities.

### *1.3.2 Vector Storage and Efficient Retrieval*

To facilitate scalable storage and retrieval of high-dimensional embeddings, a vector database (ChromaDB) is used. ChromaDB provides efficient handling of embeddings, and structured data management, which are critical for subsequent analysis.

### *1.3.3 Semantic Similarity Measurement*

To quantify relationships between embeddings, cosine similarity is employed. This metric evaluates the semantic closeness of different language

samples, allowing for an initial exploration of multilingual similarity patterns.

### *1.3.4 Dimensionality Reduction and Visualization*

Since the embeddings exist in a high-dimensional space (1,024 dimensions), t-SNE is applied to reduce the dimensionality, enabling visualization of clustering patterns among languages in a 2D space. This step helps understand potential linguistic groupings and relationships within the embedding space. By analyzing the resulting clusters, the thesis provides insights into how languages relate within a shared embedding space.

## 1.4 Outline

This thesis is structured as follows:

In chapter 2, the necessary background knowledge, topics and terminologies are explained, discussing the evolution of natural language processing, embedding models, and visualization techniques like t-SNE. This chapter guides the reader through the concepts required to understand this thesis. Additionally, related works and the existing literature on embeddings, and their relation to this thesis are presented. The chapter discusses the methodologies and findings of previous studies.

In chapter 3, the methodology and techniques used in this thesis are explained, so that a general idea of the approach can be obtained.

Chapter 4 builds on this section, and delves deeper into the technical details of the implementation. Every step is explained in detail, from data collection to visualization.

In chapter 5, Evaluation and Discussion, the results of the implementation are presented and discussed in detail, and the implications of findings are explored. The chapter explores the findings and insights gained from the analysis of multilingual embeddings, and explains the results and relations observed in the t-SNE plots.

Finally, in chapter 6, the thesis is summarized, and the contributions and findings are revisited. The chapter also provides a reflection on the research process and the lessons learned during the thesis work.

# 2 Background and Related Work

## 2.1 Natural Language Processing (NLP)

Natural language processing (NLP) focuses on understanding and processing human language, and it aims to develop techniques for computers to understand and analyze natural language text. NLP tasks include sentence parsing, text categorization, machine translation, question and answer systems, sentiment analysis, and much more [8].

### *2.1.1 Text Mining*

To be able to understand textual data, text mining and its applications should be explored. Preprocessing methods are needed to produce consistent multilingual inputs when generating embeddings. This section provides an overview of text mining, text preprocessing, and challenges related to NLP, which are helpful in understanding the subsequent sections.

As a broad field that includes information retrieval (IR) and NLP, text mining has a range of different applications, such as:

- Unsupervised Learning (e.g., document clustering)
- Supervised Learning (e.g., sentiment analysis)

or

- Language-dependent Tasks (e.g., machine translation)
- Language-independent Tasks (e.g., keyword extraction)

The goal is to turn data into a format such that data science and machine learning techniques (e.g., clustering, classification, prediction, etc.) could be applied to [9].

*Example of Applications*

- **Document Clustering:** A form of unsupervised task, with the goal of grouping documents based on a text, topic, or content similarity.
- **Document Classification:** A form of supervised task, which aims to predict a specific label for a document based on its content.
- **Keyword Extraction:** For instance, to identify significant terms in a text.

- **Machine Translation:** Automatically translating from one language to another.

*Some Unique Fields and Challenges Related to NLP*

- **Named Entity Recognition (NER):** To find and recognize named entities in a text and label them according to the contextual information: For instance, "person", "location", etc. [10]
- **Part-Of-Speech (POS) Tagging:** To label words with their corresponding part of speech. For example, "noun", "verb", "adjective", etc.
- **Coreference Resolution:** To identify words and expressions that refer to the same entity in a text. For example, what "he" refers to, in an example sentence: "I had dinner with Sam, because he invited me."
- **Ambiguity:** In natural languages, context matters, since natural languages are complex and require substantial preprocessing. For instance, a complex sentence like "Buffalo killed a buffalo in Buffalo." is rather difficult for preprocessing.
- **Homonyms:** Some words could have several different meanings, even though all the variations are written with the same syntax. (E.g., Sign, Firm, Tie, Fly, Watch, etc.)

Some more challenges are prepositional attachment, anaphora resolution, presupposed and hidden information, etc.

### *2.1.2 Text Preprocessing*

The first challenge of text mining is to go from unstructured data (text) to structured data (numbers). Texts are usually unstructured [9], requiring transformation into structured data through methods such as tokenization [11] and stop-word removal [12]. Or that the "length" of a single unit of information may vary dramatically. In order to make text "processable", a text has to go through multiple steps of transformation.

Initially, a corpus needs to be selected from the data sources. Then, the corpus needs to be divided into pieces that are consistent, e.g., words, sentences, paragraphs, tweets, posts. The pieces of text in the corpus are often called documents (regardless of nature and size, e.g., tweets, e-mails, webpages, articles, chapters, etc.) [9, 13].

An annotated corpus is a corpus in which the documents or elements within the documents have been annotated with additional information to work as a training set for a specific application [14]. These texts are usually annotated by hand. The early innovations in NLP were only possible thanks to the people who annotated tens of millions of texts by hand back in the 80s [13–15].

After a corpus is obtained or generated, the text has to go through a preprocessing phase. This step is where things like tokenization, stop-word removal, and token normalization (stemming or lemmatization) happen.

### *2.1.3 Tokenization*

Tokenization means splitting a text into similar units called "tokens". The units chosen for tokenization are very task- and language-dependent, but usual units are words, characters, ideograms, phonemes, syllables, sentences, phrases, clauses, and more [11, 16].

In tokenization, languages differ from each other greatly. It is important to note that tokenization is not trivial, not even into words. There are specific tokenizers, and this process is usually specifically designed ad hoc for a certain task, since it is very application-dependent [11].

#### *Stop-Word Removal*

Stop-word Removal refers to removing words that are not informative [12]. For example, articles, prepositions, pronouns, such as 'the', 'as', 'in', 'me', 'you', 'which', 'on', etc.

A list of stop-words is called a stop list. Stop lists are language-dependent. Here are common examples of what a stop list would commonly contain in the English language:

- **'Be' and 'have' verbs:** *'is', 'are', 'am', 'was', 'have', 'has', etc.*
- **Articles:** *'the', 'a', 'an', etc.*
- **Auxiliary verbs:** *'will', 'should', 'would', 'shall', 'must', etc.*
- **Prepositions:** *'in', 'to', 'from', 'through', 'of', 'by', 'on', etc.*
- **Question words:** *'who', 'what', 'which', 'where', 'when', 'how', 'why', etc.*

#### *Stemming*

Stemming means the process of reducing words to their word stem, base, or root form using different methods, for example, removing suffixes [17].

For instance: *"compute", "computer", "computers", "computing", "computational"* $\mapsto$ *"comput"*

#### *Lemmatization*

Unlike stemming, lemmatization applies vocabulary and morphological analysis, aiming to remove inflectional endings and return the base or dictionary form of a word, known as the lemma [18].

For instance: *"compute", "computer", "computers", "computing", "computational"* $\mapsto$ *"compute"* (lemmatization) vs *"comput"* (stemming).

*Token Normalization*

Stemming and lemmatization are the main forms of token normalization. The process of transforming tokens to make them comparable is called token normalization [17].

Some examples of other forms of normalization include:

- **Case-folding:** Refers to the process of converting everything into lowercase format.
- **Alternative spelling:** Means considering different spellings (e.g., British vs. American) *"color"* and *"colour"*.

## 2.2 Autoencoders and Latent Space

This thesis builds on the idea of a 'latent space' by investigating how multilingual transformers encode multiple languages into a shared embedding space, thus uncovering cross-lingual relationships. To understand these concepts, this section provides an overview of autoencoders, latent space, embeddings, and vector databases. Autoencoders not only demonstrate how data can be compressed into a reduced representation, but they also illustrate how information from diverse languages can be unified in a single shared latent space. This is particularly useful in cross-lingual settings, as similar linguistic structures or semantic relationships (regardless of language) tend to occupy nearby regions in that space, enabling effective multilingual analysis and retrieval.

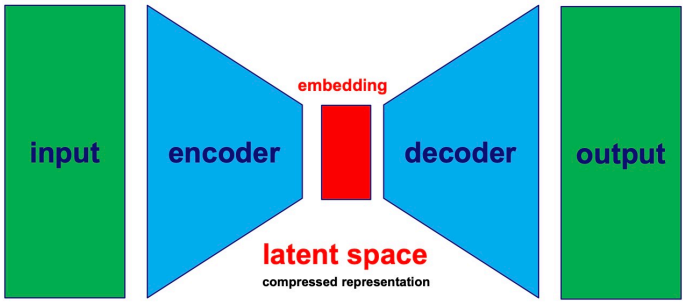



Figure 2.1: Architecture of an Autoencoder and representation of Latent Space

Imagine a neural network, where the input and output layers are both of dimension $N$, and a hidden layer of dimension $K$ with $K$ being significantly smaller than $N$. This neural network has one training task: the goal is to reconstruct the input. This is used to create a compact representation for a word, document, image, etc. [17]

The encoder encodes the input into a vector with fewer dimensions in an approach to "compress" it and learn a compressed representation of a

certain input in a latent feature space or embedding space. The output of the hidden layer (latent space) is that compressed representation that was sought. The compression ratio is $N/K$ [17].

These neural networks are called **autoencoders** [19], and present an example of how it is possible to automatically learn a representation of the data. Learning a (often smaller) representation of data is called **embedding**. When applied to text, it is referred to as **word embeddings**. It is possible to split an autoencoder into the encoding and decoding parts after the training.
As it can be seen from 2.1, the model consists of two parts:

- **Encoder:** Reduces the input data to a smaller, compressed representation (latent space).
- **Decoder:** Reconstructs the original input from this compressed representation.

### *2.2.1 Latent Space*

Latent space is the middle layer between the encoder and decoder, and it holds the compressed representation of the data, also called an embedding. This representation captures essential features of the data in a more efficient and compact form. Since it captures the core structure of the input, it allows for efficient data representation and transformation. The latent space helps in tasks like clustering, classification, or generating new data based on learned features, and it is where the input data is mapped into a simpler form that retains the most important information needed to reconstruct the original input or perform other tasks. [5]

Formally, latent space is defined as an abstract multi-dimensional space that encodes a meaningful internal representation of externally observed events. Samples that are similar in the external world are positioned close to each other in the latent space.

#### *Latent Space in Embeddings*

In embeddings, latent space refers to a mathematical space where data points (in this case, text representations) are embedded as vectors, capturing meaningful patterns, features, or relationships in a compressed and abstract form. This representation is "latent" because it is not directly observable, but derived through machine learning models like embedding models (e.g., transformers). The goal of embedding text into a latent space is to preserve the semantic relationships between the inputs in a way that can be efficiently processed by a machine [5].

In NLP, word embeddings are numerical representations of words so that similar words have close representations. Thus, word embeddings lie in a latent space where every word is encoded into a low-dimensional semantic vector [5].

*Shared Latent Space*

A shared latent space specifically refers to a single, unified representation space where data from multiple languages is embedded. In multilingual models, this shared space allows representations of texts from different languages to coexist and ideally align based on their semantic similarities [20]. For example:

- Words or phrases with similar meanings across languages (e.g., "cat" in English, "gato" in Spanish, and "Katze" in German) should have embeddings that are close to each other in the shared latent space.
- Language-specific characteristics or clusters may still emerge, but they exist within the same overarching embedding structure.

A shared latent space is central to multilingual NLP because:

1. **Cross-Language Transfer:** Models trained in one language can transfer knowledge to others when their embeddings share common structures, enabling tasks like zero-shot translation or multilingual sentiment analysis.
2. **Cluster Analysis:** Analyzing how languages are distributed or cluster in this space reveals insights into linguistic relationships (e.g., similarities between language families or imbalances caused by training data).
3. **Equity:** Understanding shared latent spaces helps in mitigating biases, ensuring underrepresented languages are not poorly represented or marginalized.

Latent space is the abstract representation of text data, and a shared latent space reflects how multiple languages are encoded together in a way that captures their interrelations and individual nuances [20].

### *2.2.2 Embeddings*

Embeddings are numerical representations of entities such as text, images, and audio, specifically designed for machine learning models and semantic search algorithms. These representations encode the features, attributes, and categorical associations depending on similarities and differences of the features of the mentioned objects into a numerical format suitable for processing [5].

This allows machine learning models to identify similar entities. For instance, in NLP, given a text or a document, a machine learning model could use embeddings to identify similar words or sentences. Embeddings allow models to understand the relationships and semantics between words and other objects. Technically, embeddings are vectors created by machine-learning models (autoencoders) to capture meaningful data and better representations about objects and entities [5].

Embedding is the process of creating vectors using deep learning methods. Embeddings that are close to each other, can be considered similar [5].

### *2.2.3 Vector Databases*

In machine learning, using vectors makes finding similar entities possible. It is simpler to find two vectors that are close together in a vector database, rather than understanding similarities between two different entities in other forms. A typical vector database for a deep learning model is composed of embeddings. In other words, a vector database is a collection of vectors (data) stored as numerical representations. They make it easier for machine learning models to remember previous inputs, and this enables machine learning to be used in searching and text generation. Instead of identifying exact matches, vectors allow the data to be identified based on similarity metrics. This allows the model to understand the data in a rather contextual way [21].

Each vector in a vector database corresponds to an object or item. In NLP, that is usually a word, sentence, or a document. In other areas, it could be an image or any other piece of data. These vectors are usually very complex, meaning that the representation and location of each object is defined along very high dimensions [22].

Vector databases are used by similarity and semantic searches for clustering similar and possibly relevant items together. This can be used in many different applications. Some real-world examples are: to recommend similar products in an online shop, to suggest songs or movies in entertainment platforms, etc. [21, 22]

The ability to find relevant items of data has enabled machine learning (and deep learning) models to do complex cognitive tasks. Large Language Models (LLMs) also rely on the contextual analysis of text made possible by vector databases. By identifying relations between words, sentences, and documents, LLMs can understand natural language and generate text [21].

## 2.3 Attention Mechanism and Transformers

Since the chosen embedding model in this thesis relies on the transformer architecture, the attention mechanisms directly influence how multilingual texts are encoded, and thus shape the shared latent space. This section provides an overview of attention mechanisms and transformers, which are fundamental to understanding how multilingual embeddings are created.

The concept of attention was first studied in the context of computer vision to recognize some objects in an image [23]. It was shown to be effective in selectively attending to parts of the source sentence while

generating the target sentence. Other real-world applications such as automatic speech recognition in listen, attend, and spell model (LAS) and NLP showcase the practicality and effectiveness of attention mechanisms [24–26]. In 2014, attention was used to highlight specific parts of an image that relate to a desired output [23].

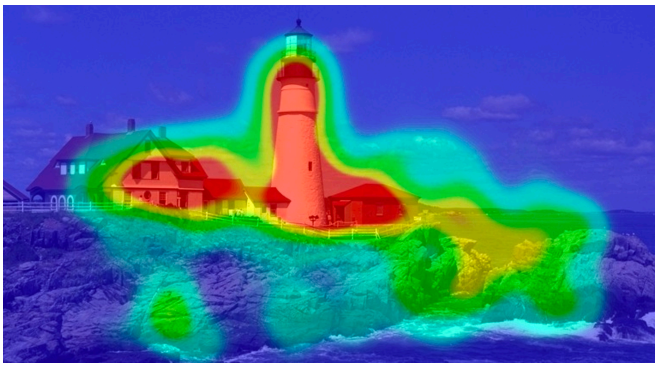


Figure 2.2: Attention mechanism used in computer vision to recognize buildings in the image [27].

In Fig. 2.2 there are two buildings. The task on hand is object detection, where the objective is to recognize buildings. What the figure tries to demonstrate here is a heat map, that is overlaid on top of the image. This heat map shows the pixels or the region where buildings are more likely to be in the red part.

If some network is trained to do some classification, an interesting question is whether the network comes up with the right class when it outputs a class. For instance, the network could also be asked to highlight where the building is, if it claims that there is indeed a building in the image. This could be a decent way to validate and understand what it thinks a building is and whether or not it is correct.

Attention can be used to identify which pixels correspond to the concept of a building, and the resulting heat map indicates the attention weights for those pixels. In this scenario, one can envision an attention mechanism that spans the entire image, assigning weights to every pixel. The model would then determine which pixels carry semantic significance or embeddings aligned with the target object to be recognized.

### *2.3.1 Attention in Natural Language Processing*

In NLP (natural language processing) and NMT (neural machine translation) the work about machine translation was introduced [24], where the decoder could be used to effectively peek or look back at what the input sentence was so that it would not lose track of what it is translating and it does not have to remember the sentence completely. This was an

important breakthrough that enabled dealing with sentences of arbitrary (and more importantly, very long) length [24].

In 2017 researchers showed that attention can be used to develop general language modeling techniques [25]. Here, language modeling refers to the idea of developing a model that would predict the next word in a sequence. It could generate words and then if there is a word missing somewhere in a sequence, it could also recover that missing word. In essence, a language model is a system that can predict or recover missing words in a sequence. Many tasks in NLP can be formulated as language modeling problems. For instance, translation can be viewed as a sequence prediction task: the first part of the sequence is a sentence in the source language, and the model then continues the sequence in the target language by predicting the next words. Thus, many tasks in NLP can be framed as variants of language modeling. Vaswani et al. demonstrated that an architecture using almost exclusively attention blocks could be designed; they named this architecture 'the Transformer'.

$$\text{attention}(q, k, v) = \sum_i \text{similarity}(q, k_i) \times v \tag{2.1}$$

### *2.3.2 Computing Attention*

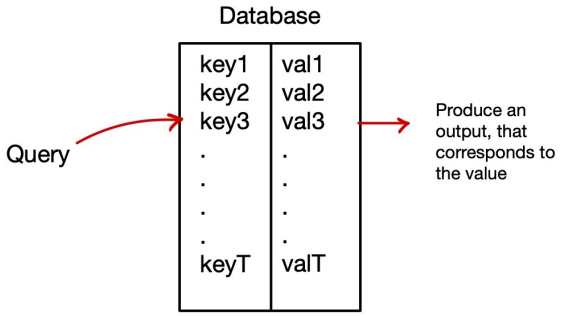



Figure 2.3: Attention mechanism mimics the retrieval of the value $v_i$ for a query $q$ based on a key $k_i$ in the database [27].

Attention can essentially be thought of as being some form of approximation of a selection that would be done in a database. As shown in Fig.2.3, in a database, to retrieve a value based on some query and some key, an operation can be done, where a query is used to identify a key that aligns well and outputs the corresponding value. Attention can effectively be conceived as imitating the retrieval process but in a more probabilistic manner. What is measured in (2.1) is the similarity between the query $q$, and each key $k_i$. Then this similarity is going to return a weight and then a weighted combination of all the values in the database will be produced as an output.

In the following, (2.1) will be decomposed into distinct steps, and each step will be explained in more detail individually.

### Computing the Similarity

Attention is used to determine which parts of the input sequence are most relevant to a given query. In this case, the query $Q$, key $K$, and value $V$ matrices represent different aspects of the input sequence. The query matrix, $Q$, represents the current query or task that the model is trying to perform, the key matrix, $K$, represents the elements in the input sequence, and the value vectors in the matrix $V$ are weighted using the weights resulting from the softmax operation, so $V$ represents the values or output that the model wants to compute.

In (2.2) $s$ corresponds to the similarity of the query $q$ to each key vector $k_i$, where:

$$s_i = f(q, k_i) \tag{2.2}$$

There are many functions which can be considered for $f(q, k_i)$. Including the dot product: $q^T k_i$, or the scaled dot product: $q^T k_i / \sqrt{(d_k)}$, where $d$ is the dimensionality of each key, and is normally used to avoid the dot product from becoming too large. As shown in (2.2) in the first step of computing attention, the scaled dot product of the query and key matrices, $QK^T/\sqrt{d_k}$ is calculated. This results in a matrix that has the same number of rows as the query matrix and the same number of columns as the key matrix.

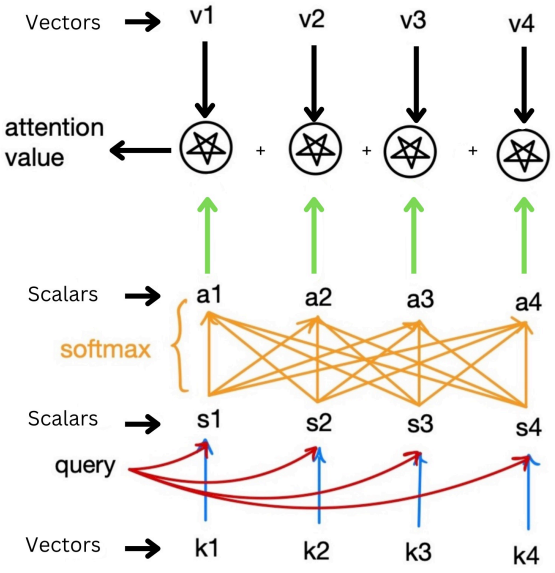



Figure 2.4: Computing Attention [27]

### Computing the Weights

In the next step, the weights need to be computed. To do that, the resulting matrix $s_i$ is passed through a softmax function to compute the attention weights $a_i$, as shown in (2.3).

$$a_i = \frac{\exp s_i}{\sum_j \exp s_j} \tag{2.3}$$

The softmax function maps the similarity values $s_i$ to a probability distribution, where each element in the output matrix represents the probability that the corresponding key element should be considered when computing the output.

### Computing the Final Attention Value

Finally, the attention weight $a_i$ is multiplied by the value vectors $v_i$ to obtain the final attention output, as shown in (2.4). This output is a weighted sum of the values, where the weights are determined by the similarity between the keys and the query.

$$\text{attention value} = \sum_i a_i v_i \tag{2.4}$$

As shown in Fig. 2.4 this is a general scheme, where a query $q$, and some keys $k_i$ are given, and the goal is to produce an output where the output is a linear combination of the values $v_i$, where the weights $a_i$ come from some notion of similarity between they query and the keys.

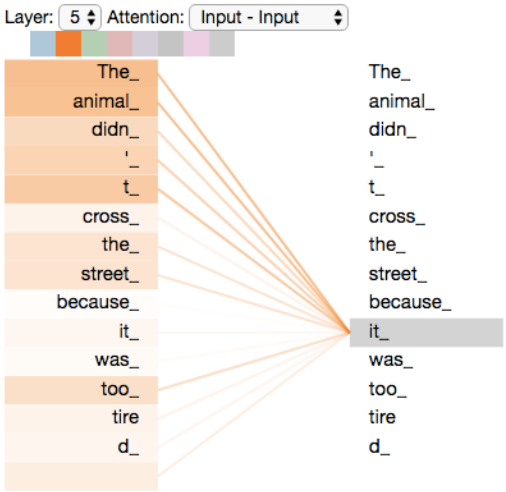



Figure 2.5: Part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it" [28].

### *2.3.3 Self-Attention*

Self-attention is when the attention mechanism is used for a sequence to attend to different parts of itself when processing each element [29]. It takes into consideration the relationship among words within the same sentence. The model attends to different parts of the sequence while processing each element [29].
Fig 2.5 is an example to better visualize the model's ability to attend to different parts of itself. The animal didn't cross the street because it was too tired.

In Fig. 2.5 it can be observed, the word "it" is mostly focused on the word "The Animal". It understands what "it" refers to in the sentence.

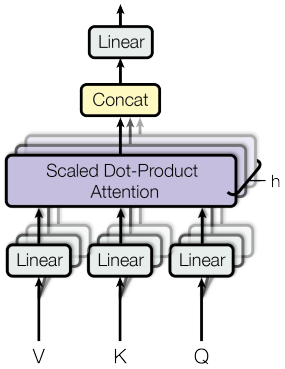



Figure 2.6: Multi-Head Attention [25]

#### Multi-Head Attention

Self-attention could be further refined and expanded by adding multiple attention heads to it. This allows it to attend to different elements simultaneously and expands the model's ability to focus on different positions at once [25]. This involves splitting the input into multiple heads (subspaces), effectively creating multiple sets of $Q$, $K$, and $V$ vectors, and applying attention mechanisms independently to each head, rather than using a single attention head. This now allows the model to attend to different parts of the sequence, and capture different aspects or patterns in the data simultaneously [25]. The outputs from all heads are then concatenated using (2.5) and linearly transformed to produce the final attention-weighted representation, as demonstrated in Fig. 2.6.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \tag{2.5}$$

where $\text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ [25]

When revisiting the previous example in Fig. 2.7, this time the multi-head attention will be taken into use instead of single-head self-attention.

It can now be seen, with one attention head (orange) the word ”it” still focuses on ”The Animal” and the other head (green) focuses on ”tired”, which means, it now knows the word ”it” refers to an animal, which is tired.

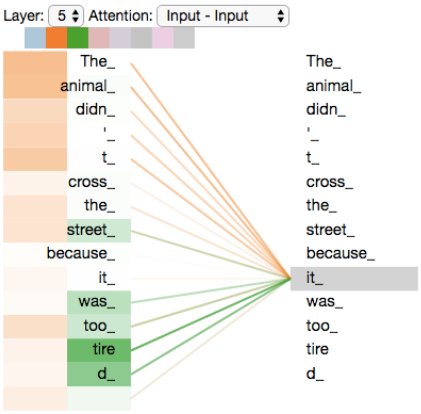



Figure 2.7: When encoding the word ”it”, one attention head is focusing on ”the animal”, while another is focusing on ”tired” [28].

### Masked Multi-Head Attention

The next concept is called masked self-attention, which is used to prevent the decoder from ”peeking” into the future when generating the target sequence. This is done by masking certain positions in the decoder's self-attention mechanism so that the decoder can only attend to positions that come before the current position in the input sequence. The idea of having

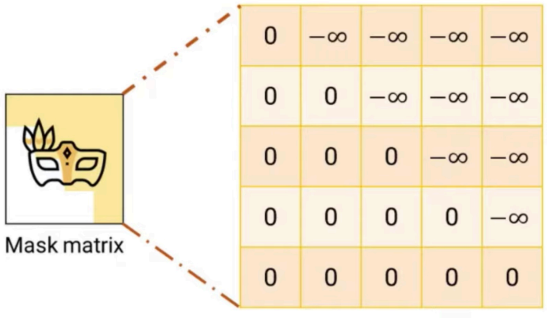



Figure 2.8: The mask matrix is an upper triangular matrix with entries above the main diagonal set to $-\infty$.

masked attention is that some of the values should be masked, meaning that the probability should be nullified, so that they do not create any combinations. So for instance, in the decoder when the output is produced in a machine translation task, there exists a sentence (for example) in

English, and the goal is to translate that into French. The words are produced in French, and when a word is produced, then it is acceptable for that word to depend on the previous words in the translation, because they are being generated sequentially, but that word should not depend on the future words, because they have not been produced yet. So, what needs to be done is to essentially change the attention mechanism, so that the links that would create dependencies on words that have not been generated yet would be nullified or effectively removed.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.6}$$

$$\text{Masked attention}(Q, K, V) = \text{softmax}(\frac{QK^T + M}{\sqrt{d_k}})V \tag{2.7}$$

where M is a mask matrix of 0's and $-\infty$'s [25]

This is what is call masked multi-head attention. The main difference with normal multi-head attention is that in the attention mechanism, a softmax function is computed according to (2.6), but now, according to (2.7), with masked attention, a mask is going to be added, that will effectively produce some probabilities, that are zero for the terms that should not be attended to, because they would be future terms. In (2.3) if there was a mask, which is as shown in Fig. 2.8, an upper triangular matrix of 0's and $-\infty$'s, when calculating $\exp(-\infty)$, the result is 0. So this has the effect of ensuring that the probabilities of certain items here are going to be 0, and this will have the same effect as removing connections.[27]

### *2.3.4 The Transformer*

The Transformer architecture is encoder-decoder based. The encoder encodes the input sequence into a representation, and the decoder has to generate the target sequence from this representation. Both the encoder and the decoder consist of self-attention and feed-forward neural network layers, which can be seen in Fig. 2.9.

#### The Encoder

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sublayers: a multi-head attention and a simple position-wise fully connected feed-forward network, which is used to transform the output into a new representation, and is applied to each position separately and identically. Other than these, a residual connection is employed around each of the two sublayers, followed by layer normalization. Since the Transformer was originally proposed for translation tasks, the inputs are a series of words. Because there are no words when calculating probabilities, the words should be converted into numbers. There are different ways to

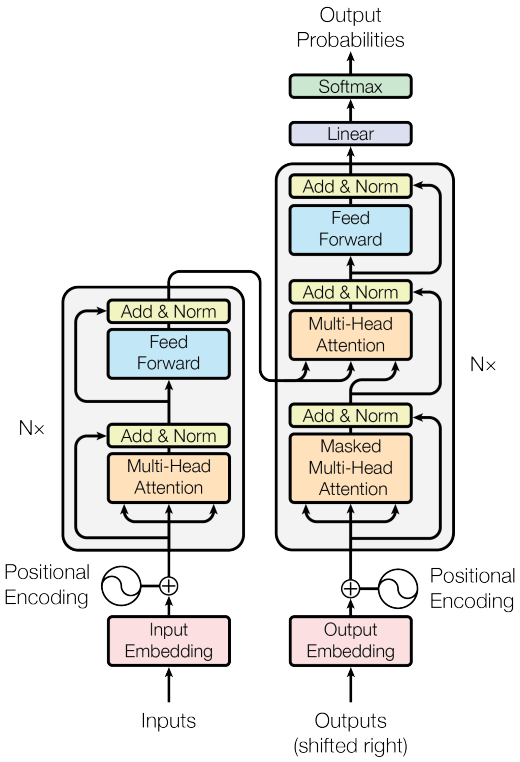



Figure 2.9: Transformer Model Architecture [25]

do so. The computer could be given a dictionary, where each number represents a word, but there is a better method, called word embedding. Word Embedding is a method of representing words with vectors in such a way, that similar words have similar vectors. For instance, words like cat and dog should have similar vectors, but words like cat and car should have relatively different vectors, despite having similar characters. In positional encoding, the purpose is to add some information about positions before feeding the embeddings to the encoder, allowing them to differentiate between words with similar meanings but different positions in a sentence, because word positions matter a lot in sentences. A good example to show the importance of positional encoding is two sentences with the same set of words, which have completely different meanings. The sentences Mercedes defeated Red Bull and Red Bull defeated Mercedes contain the same words but convey entirely different meanings simply due to the change in word order. The attention block is followed by an add and norm layer, which can be seen in Fig. 2.10. In this layer, first, the sum of the output vector of the attention block and the input embedding vector is calculated, which was retrieved in the first step. The result is then subjected to layer normalization. Normalized data plays an important role in increasing the training speed and reducing bias. The general rule and purpose is to process the output from one attention layer in a way that better fits

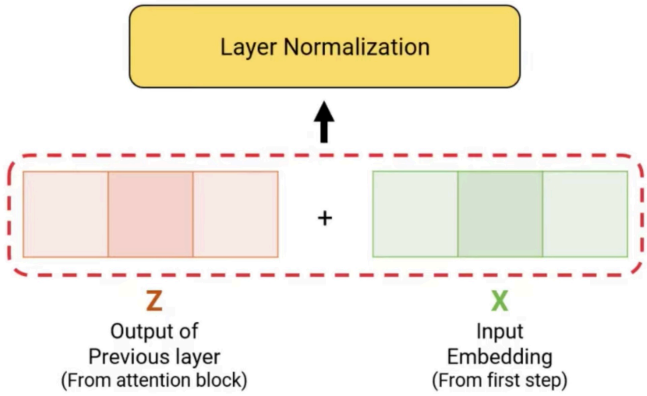



Figure 2.10: Add and Norm Layer [25]

the input for the next attention layer. The residual connections send the output of each sublayer to the add and norm block of the next layer.

#### The Decoder

The decoder is also composed of a stack of $N = 6$ identical layers. Other than that, the decoder also inserts a third sublayer, which performs multi-head attention over the output of the encoder stack. In the second attention layer of the decoder, the Key $K$ and Value $V$ data come only from the top encoder block to each and every decoder block. However, the Query $Q$ data is taken from the first attention layer of the decoder.

## 2.4 Visualization and Metrics of Similarity

Visualization is a powerful tool to understand complex data and their relationships. Through visualization it is possible to find and identify patterns, which would otherwise remain hidden in the data. In the context of machine learning, visualization is used to understand the relationships between data points, clusters, and dimensions. It is particularly useful in understanding the behavior of models and the quality of the data. One of the visualization techniques using dimensionality reduction is t-Distributed Stochastic Neighbor Embedding (t-SNE).

This thesis relies on t-SNE to visually inspect how embeddings of different languages cluster in a high-dimensional space.

### *2.4.1 t-SNE*

Visualizing high-dimensional data is usually difficult, but is an important problem, which deals with data of many different dimensionalities. t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality

reduction technique used for visualizing high-dimensional data in a lower-dimensional space [30]. In a t-SNE plot, the closer the datapoints are, the more similar they are in the high-dimensional space. t-SNE is particularly useful for visualizing relationships between data points in a complex dataset, such as embeddings [30]. This trait of t-SNE makes it useful for visualizing embeddings in a shared latent space, as it can reveal patterns that are not immediately apparent with looking at the raw data or the embedding vectors, and makes the understanding of the relationships between different languages easier.

### *2.4.2 Cosine Similarity*

A way to compute the similarity of two vectors is to check whether two vectors are pointing in roughly the same direction and to calculate the angle between them. For this purpose, cosine similarity is used, a widely employed measure in natural language processing. The angle between two vectors is calculated as shown in (2.8).

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} \tag{2.8}$$

Where:

- **A** and **B** are the embedding vectors.
- $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the norms of these vectors.

The cosine similarity value can range as follows:

- 1: Perfect alignment or maximum similarity
- 0: No similarity
- -1: Complete opposition

## 2.5 The Dataset

This thesis focuses on 24 selected languages from different families to observe how the chosen embedding model groups them in latent space. This section provides an overview of the dataset used in this thesis.

### *2.5.1 C4*

The dataset used in this thesis is an updated and advanced version of the original Colossal Clean Crawled Corpus (C4) dataset, which is a large-scale multilingual dataset containing web pages in multiple languages. The C4 dataset is a valuable resource for training multilingual language models, as it provides a diverse range of textual data across numerous languages [2]. The size of the dataset makes it useful for studying and understanding the different content and topics covered in it. It comprises approximately

750 GB of data, focusing on natural language content, and has undergone extensive cleaning and preprocessing to ensure high quality and consistency [2]. It is primarily intended for pretraining language models and word representations. The largest version of publicly available C4 dataset consists of the following: 305GB cleaned JSON files, 2.3TB uncleaned files and 15GB of real news like data. This dataset is deprecated [2].

### *2.5.2 mC4*

The multilingual C4 (mC4) dataset is an extension of the C4 dataset that includes natural text in 108 languages, also sourced from the Common Crawl web scrape. This multilingual variant allows for analysis on the difference between a wide variety of topics and different languages. This extension is significantly larger than the original version, with over 13.5 TB of data, making it a rich resource for multilingual NLP research [31].

### *2.5.3 AllenAI mC4*

This version includes all the same parts of the C4 dataset, but also includes 9.7TB (108 languages) multilingual data. This is the most up-to-date version of the dataset [32]. In this thesis, 24 languages have been chosen from the dataset. When choosing these languages several factors were taken into consideration. The goal was to choose languages from both similar and different language families to see how they would cluster in the shared latent space. Also, the languages chosen should have been availble in the dataset, and compatible with the embedding model used. Another criterion was that some languages should have a lot of data available in the dataset, when others do not, so it was important to choose languages with variable data availability. After many considerations, the following 24 languages were chosen: English (en), German (de), Dutch (nl), Swedish (sv), Norwegian (no), Danish (da), Spanish (es), French (fr), Italian (it), Portuguese (pt), Romanian (ro), Russian - Latin (ru, ru-Latn), Polish (pl), Czech (cs), Farsi (fa), Irish (ga), Finnish (fi), Hungarian (hu), Chinese - Latin (zh, zh-Latn), Arabic (ar), former Hebrew (iw), Tamil (ta), Telugu (te), Japanese - Latin (ja, ja-Latn). Table 2.11 demonstrates the selected languages (with their corresponding fammilies, and language groups).

This covers a wide range of language families, including Indo-European languages like Germanic, Romance, Slavic, Indo-Iranian, and Celtic languages, as well as Uralic, Dravidian, Japonic, and Afro-Asiatic languages. The dataset is expected to provide a diverse set of linguistic features and relationships for analysis. The Germanic languages (English, German, Dutch, Swedish, Norwegian, Danish) are quite similar to each other. The Romance languages (Spanish, French, Italian, Portuguese, Romanian) also share many similarities. On the other hand, Mandarin Chinese from the Sino-Tibetan family is quite dissimilar from the Indo-European

Table 2.11: List of languages chosen, grouped by language family and subgroup.

| **Language Family** | **Subgroup** | **Languages (Code)** |
|---|---|---|
| **Indo-European Languages** | Germanic Languages | English (`en`), German (`de`), Dutch (`nl`), Swedish (`sv`), Norwegian (`no`), Danish (`da`) |
| | Romance Languages | Spanish (`es`), French (`fr`), Italian (`it`), Portuguese (`pt`), Romanian (`ro`) |
| | Slavic Languages | Russian (`ru`), Russian-Latin (`ru-Latn`), Polish (`pl`), Czech (`cs`) |
| | Indo-Iranian Languages | Farsi (Persian) (`fa`) |
| | Celtic Languages | Irish (`ga`) |
| **Uralic Languages** | | Finnish (`fi`), Hungarian (`hu`) |
| **Sino-Tibetan Languages** | | Chinese (`zh`), Chinese-Latin (`zh-Latn`) |
| **Afro-Asiatic Languages** | | Arabic (`ar`), former Hebrew (`iw`) |
| **Dravidian Languages** | | Tamil (`ta`), Telugu (`te`) |
| **Japonic Languages** | | Japanese (`ja`), Japanese-Latin (`ja-Latn`) |

languages. Similarly, Dravidian languages (Tamil, Telugu) and Afro-Asiatic languages (Arabic, Hebrew) are also quite different from the Indo-European languages.

## 2.6 BAAI bge-m3 Embedding Model

The embedding model used in this thesis is the BAAI bge-m3 model [33]. This model has been trained on a mixture of unsupervised, supervised, and synthetic data across multiple languages and it is a multilingual model that can generate embeddings for 108 languages. The model is based on

the transformer architecture, which is known for captureing long-range dependencies in text data using attention mechanism. It is capable of processing inputs of different lengths, from short sentences to long documents up to 8192 tokens [33]. In the bge-m3 model, the dimension count, which is the size of the embedding, or the number of features that each token is represented by is 1024. This value is important because it shows the capacity of the model to encode the information in each vector [33].

## 2.7 Related Work

### *2.7.1 Exploring Cross-Domain Semantic Similarity through Embedding Models*

Yildiz et al. propose in the paper Investigating Continual Pretraining in LLMs, through embedding models, to find semantic similarity across domains [34]. This work explores the application of continual pretraining in LLMs for cross-domain knowledge transfer evaluation. The study has utilized the Massively Multi-Domain Dataset (M2D2) [34, 35], a hierarchically structured corpus containing 236 domains sourced, of which 159 were chosen in this study (excluding domains exceeding 5GB of data) from platforms such as Wikipedia and Semantic Scholar. The main purpose was to measure the adaptability of LLMs when exposed to varying domain contexts, while retaining previously learned knowledge and increasing forward and backward transfer abilities.

The authors implemented Sentence-BERT (SBERT) as their embedding model used in the experiments on domain similarity [34, 36]. SBERT, is a variant of BERT for sentence-level tasks, which authors used to encode task embeddings by processing 10,000 samples from each domain, and another 50,000 samples from OpenWebText. Further analysis was made on these embeddings using cosine similarity to quantify the relationship of various domains. A t-SNE plot is provided to visualize the embeddings, which allows identifying semantically similar domains. The results indicate that embedding-based representations are able to capture both domain-specific and cross-domain semantic relations. This helped in evaluating continual pretraining strategies, especially when combined with analyses of domain ordering and knowledge transfer.

### *2.7.2 Continual Pretraining and Domain Adaptation in LLMs*

The work [34] is grounded in continual learning and domain adaptation. This current study is an extension of [37], which tested the transfer capabilities of a RoBERTa model on four domains. More research, such as [38], has shown continually pre-trained RoBERTa and BERT to be robust in avoiding catastrophic forgetting, when tested on downstream tasks. On another note, a soft-masking strategy for continual pretraining

in RoBERTa was found to be effective when applied across a variety of tasks [39].

While [34] refers to such studies in order to introduce the context of continuous pretraining, their main contribution is evaluating cross-domain similarity and knowledge transfer. Their approach, by using SBERT, cosine similarity, and t-SNE visualization, is what separates their work from previous research that mostly dealt with task performance and domain-specific fine-tuning. Their approach not only puts the focus on the adaptability of embedding models like SBERT to cross-domain scenarios, but also explains how domain ordering impacts knowledge retention and transfer in LLMs.

### *2.7.3 Relation to This Work*

The methodologies in the work [34] provided the main inspiration for this thesis. Their use of embedding models for domain similarity analysis and the application of cosine-similarity metric and t-SNE for simple and understandable visualization directly shaped and influenced this thesis to explore lingual proximity. However, this thesis has a different focus and application. While their work analyzes cross-domain relationships, this thesis looks into the proximity of languages through the embeddings created by multilingual models. The main hypothesis for this thesis states that embeddings can uncover linguistic relations and family cluster, independent of semantic meaning. This extends the idea of encoding structural and semantic properties in embeddings to a whole new level, studying language families and their relations.

Another difference refers to the analytical methods. While [34] were interested in domain similarity using cosine similarity metrics, this thesis adapts these ideas to measure linguistic proximity, namely how close or far are embeddings of different languages. By adapting their methods to a multilingual context, this thesis uses their results while addressing a new research question.

### *2.7.4 Methodological Similarities*

This study shares some very important similarities with the study [34] Specifically:

Embedding Models: Both studies use embedding models, that is, SBERT used in their study and the BAAI bge-m3 multilingual embedding model used in this thesis for representing and encoding textual data in high-dimensional vector spaces.

Cosine Similarity: Although cosine similarity was utilized to measure relationships between embeddings in different domains [34], in this thesis it was not considered an impactful metric because it consistently yielded very high similarity values. This thesis, therefore, relied more on t-SNE

visualizations to investigate and point out linguistic differences and similarities.

t-SNE Visualization: While t-SNE was a valuable tool for visualizing embeddings and clarifying results in their study, it played an even more important role in this thesis by being the main way to analyze and highlight linguistic differences and similarities.

### *2.7.5 Brief Summary*

The study [34] is a foundational and essential reference for the study of embedding-based similarity analysis for this thesis, particularly for the relationships between domains. While their study focuses more on domain-specific knowledge transfer and continual pretraining, this thesis extends the idea to a linguistic domain to investigate the relations of languages and the representation of language families in a shared embedding spaces. This approach emphasizes the flexibility of embedding models in capturing both semantic and structural features, thereby providing deeper insight into the dataset and its contents.

# 3 Conceptual Approach

## 3.1 Methodological Framework

A concrete plan and approach are necessary for working toward the goals of this thesis and assessing the validity of the hypothesis. This chapter explains the conceptual approach to exploring linguistic proximity, and conducting the required analyses. Each step of the approach is explained in detail, and the reasoning behind the choices made is provided.

Fig. 3.1 demonstrates the approach to exploring linguistic proximity. The main idea is to select a multilingual dataset, embed the texts using a multilingual model, store the resulting vectors in a vector database, measure similarity with cosine similarity, and apply t-SNE for dimensionality reduction and visualization to be able to conduct the analysis in an easier and more understandable manner.

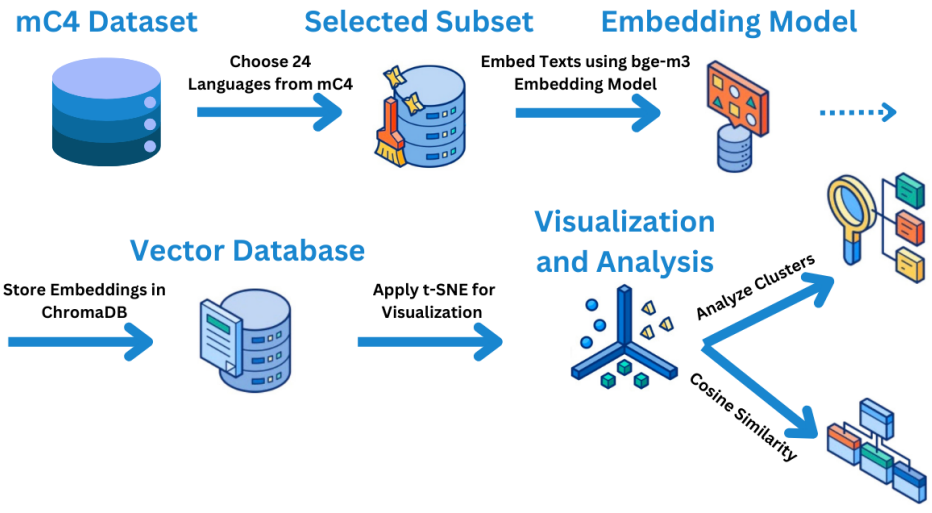



Figure 3.1: Conceptual approach to exploring linguistic proximity.

## 3.2 The Dataset

The main idea for the implementation is to choose the dataset first. Some criteria need to be followed to select a suitable dataset. The dataset should contain many different languages. It should also be formatted in a way that allows efficient processing and compatibility with the embedding model.

After considering the requirements, the mC4 dataset was chosen. The mC4 dataset is a multilingual variant of the C4 dataset. It consists of natural text in 101 languages drawn from the public Common Crawl web scrape. To choose the languages from the dataset, a limited number of languages had to be selected.

The idea was to choose languages from both similar and dissimilar families. The resource availability of languages correlates with the quality of the results, with more resources leading to better outcomes. This introduces the challenge of choosing languages with different amounts of resources available. Fig. 3.2 shows the dataset sizes for the 24 chosen languages grouped by language families and subgroups.

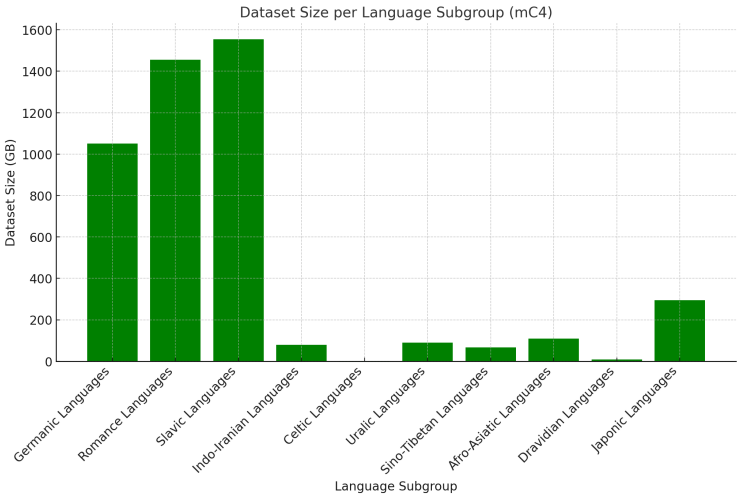



Figure 3.2: Dataset sizes, grouped by language families and subgroups [31].

## 3.3 The Embedding Model

The suitable model should be multilingual to be able to handle multiple languages and be able to generate embeddings for them. It should also be able to handle the different scripts and characters of the languages. By doing this, the essence of the languages can be captured in the embeddings, which are essentially comparable vectors, and mathematical operations can be performed on them.

Based on the mentioned points and the embedding model rankings available on Hugging Face [40], it was decided to continue with the multilingual BAAI bge-m3 model [33]. Table 3.3 provides an overview of the model's specifications and features.

Table 3.3: Specifications and specialties of the bge-m3 model.

| **Feature** | **Description** |
|---|---|
| **Model Name** | bge-m3 |
| **Developer** | Beijing Academy of Artificial Intelligence (BAAI) |
| **Dimensions** | 1,024 |
| **Maximum Sequence Length** | 8,192 tokens |
| **Multi-Functionality** | Supports dense retrieval, multi-vector retrieval, and sparse retrieval |
| **Multi-Linguality** | Processes text in over 100 languages |
| **Multi-Granularity** | Handles inputs ranging from short sentences to long documents up to 8,192 tokens |
| **Training Approach** | Unified fine-tuning encompassing dense, sparse, and multi-vector (ColBERT) retrieval methods |
| **License** | MIT License |
| **Source** | Hugging Face Model Card |

## 3.4 Storing the Embeddings - The Vector Database

The next step is to find the most efficient and effective way to save these embeddings locally. That is, a vector database. But there are several different vector databases available, with their own advantages, and disadvantages. Among the available options (e.g., Milvus, Pinecone, Weaviate, Qdrant), ChromaDB was chosen for its ease of use, native Python API, and compatibility with Hugging Face workflows. ChromaDB makes it very simple to save and manage embeddings from bge-m3.

## 3.5 Similarity Measurement

Cosine similarity is used to measure the similarity between two embeddings. Cosine similarity is a common metric for language embeddings because it captures the orientation (rather than magnitude) of the vectors, which translates to how semantically "aligned" two sentences or documents are. In practice, pairs or sets of embeddings are queried within ChromaDB and cosine similarity scores are computed to assess potential relationships.

Although detailed interpretation of results is explained in a later chapter, the main outlook is that higher cosine similarity scores may imply that two texts (whether in the same language or across different languages) share semantic or topical overlap, which does not provide much information about the specific languages themselves, or their relations to other languages.

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} \tag{3.1}$$

## 3.6 Dimensionality Reduction and Visualization

With 1,024-dimensional embeddings, it is difficult to intuitively see inter-language relationships. To address this, t-SNE is applied to project the embeddings from their original dimensionality down to two dimensions. This helps in visualizing potential language clusters, family groups, or outlier points.

The t-SNE visualization is repeated across multiple sample sizes (for instance, 10 to 200k samples) to capture how different volumes of data might influence the relative positioning of languages. The goal is to determine whether certain clusters remain stable or if patterns become more pronounced as sample size increases.

Observing a consistent and similar clustering pattern across different sample sizes indicates that the language embeddings are capturing robust

and reproducible relationships. If the same languages consistently group together (especially those belonging to the same family), it suggests that the underlying semantic representations do not significantly depend on the sample size. Also, it is ensured that the identified clusters reflect genuine linguistic relationships, rather than artifacts of a particular sample subset.

### *3.6.1 Mathematical Formulation of t-SNE*

In this subsection, the equations and concepts underlying t-SNE are explained to provide a deeper understanding of the method's operation [30]. t-SNE operates by constructing probability distributions that model the pairwise similarities between data points in both the original high-dimensional space and the lower-dimensional embedding space.

**High-Dimensional Probability Distribution**
In the high-dimensional space, the similarity between two points $x_i$ and $x_j$ is computed using a Gaussian distribution:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \tag{3.2}$$

where $\sigma_i$ is the variance of the Gaussian centered at $x_i$. The joint probability distribution is then symmetrized as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{3.3}$$

where $n$ is the number of points in the dataset.

**Low-Dimensional Probability Distribution**
To model pairwise similarities in the low-dimensional space, t-SNE replaces the Gaussian kernel with a Student-t distribution with one degree of freedom:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}} \tag{3.4}$$

where $y_i$ and $y_j$ are the low-dimensional representations of $x_i$ and $x_j$.

**Cost Function: Kullback-Leibler Divergence**
t-SNE minimizes the Kullback-Leibler (KL) divergence between the high-dimensional and low-dimensional probability distributions:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{3.5}$$

This cost function ensures that the structure of the data is preserved when transitioning from the high-dimensional space to the low-dimensional representation.

**Gradient Descent Optimization**
To minimize $C$, t-SNE updates the embeddings via gradient descent using the following gradient:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{3.6}$$

This optimization ensures that similar points in high-dimensional space remain close together while avoiding overcrowding.

By applying these transformations, t-SNE provides an effective method for visualizing complex high-dimensional relationships in a more interpretable two-dimensional space.

## 3.7 Summary of the Conceptual Approach

In summary, the conceptual workflow is:

1. Select a multilingual dataset (mC4) and narrow it to 24 languages spanning various language families.
2. Embed the texts using the bge-m3 model, chosen for its multilingual embedding abilities and robust semantic representation.
3. Store the resulting vectors in a Chroma database, which simplifies handling of high-dimensional embeddings.
4. Measure similarity with cosine similarity to assess semantic similarity across samples.
5. Apply t-SNE for dimensionality reduction and visualization of the embedding space, experimenting with multiple sample sizes to evaluate the consistency of clustering.

This plan is a conceptual approach to exploring linguistic proximity, analyzing where these languages stand in relation to each other. The more detailed experimental procedures and their corresponding outcomes will be presented in later chapters.

# 4 Implementation

## 4.1 Downloading the Dataset

The initial step is to download the dataset locally for easier and faster access during preprocessing and running the embedding model. The dataset is downloaded from HuggingFace's dataset library. This could be done using either a python script, or a bash script using Git. The second option was chosen to benefit from Git LFS (Large File Storage) to download the dataset.

Below is the bash script used to download the dataset. The script clones the dataset repository without downloading the large files. Then, it pulls the large files for the selected languages.

```
#!/bin/bash

# Step 1: Clone the repository without downloading large files
GIT_LFS_SKIP_SMUDGE=1 git clone
    https://huggingface.co/datasets/allenai/c4
cd c4

# Step 2: Pull specific large files for multiple languages
languages=("en" "de" "nl" "sv" "no" "da" "es" "fr" "it" "pt" "ro"
    "ru" "pl" "cs" "fa" "ga" "fi" "hu" "zh" "ar" "iw" "ta" "te" "ja")
for lang in "${languages[@]}"; do
   git lfs pull --include "${lang}/*"
done
```

After downloading the dataset locally on the HPC, the embedding model could be started to process the dataset. The rest of the work is done using python scripts.

## 4.2 Dataset Structure

It is crucial to know the structure of the dataset exactly before continuing the implementation. First, the dataset is loaded into the directory `c4` and the structure of the dataset is as follows:

```
en, en.noblocklist, en.noclean, multilingual, realnewslike
```

There are five directories and one README.md file. The `en` directory contains the English dataset, `en.noblocklist` contains the English dataset without the blocklist, `en.noclean` contains the English dataset without cleaning, `multilingual` contains the multilingual dataset, and `realnewslike` contains the real news-like dataset.

Two directories are of particular interest to us: the `en` directory and the `multilingual` directory.

### *4.2.1 English Dataset*

The structure of the **`en`** directory is as follows:
It contains more than a thousand **`.gz`** Gzip compressed files. The naming convention for these `.gz` files are as follows:

```
c4-train.00000-of-01024.gz,
c4-train.00001-of-01024.gz,
c4-train.00002-of-01024.gz
```

and so on, until the validation sets start:

```
c4-train.01023-of-01024.gz,
c4-validation.00000-of-00008.gz,
c4-validation.00001-of-00008.gz
```

and so on.

So, the general naming convention is as follows:

```
c4-train.XX-of-01024.gz
```

for the training set and

```
c4-validation.XX-of-00008.gz
```

for the validation set.

### *4.2.2 Multilingual Dataset*

The structure of the `multilingual` directory is a little bit different, and as follows:

```
c4-\{lang\}.*.json.gz
```

files, where lang is the language code. For example, a training set would be

```
c4-af.tfrecord-00000-of-00064.json.gz
```

for Afrikaans,

```
c4-de.tfrecord-00000-of-02048.json.gz
```

for German, and so on. The validation sets for each language are named similarly to the training sets, but with the word **validation** added to the name e.g.,

```
c4-af-validation.tfrecord-00000-of-00001.json.gz
```

for Afrikaans.

## 4.3 Generating Embeddings from the Dataset

To process the dataset, a detailed python script is written to extract the data from the Gzip files, clean the data, and save it in a format that can be used by the embedding model. The script is written in a way that it can process both the english and multilingual data from the dataset. See 6 for the full python script.

### *4.3.1 Imports and Environment Setup*

After importing some standard python libraries (e.g., os, glob, json, uuid, etc.) along with third-party libraries such as PyTorch, Hugging Face's Transformers, Datasets and Sentence Transformers, tqdm for progress bars, and ChromaDB for persisent embedding storage, the environment variables are set.

### *4.3.2 Model Initialization*

Then the model and the built in tokenizer are initialized. The script uses the BAAI bge-m3 model with a sequence length of 512, which sets the maximum token length per input, and a maximum batch size of 2048, which acts as an initial upper bound for batching, which is adjustable based on available GPU memory. After the tokenizer and model are initialized and loaded via Hugging Face's APIs, the model is moved to GPU ('cuda') and set to evaluation mode.

### *4.3.3 Dynamic Batch Size Determination and Mean Pooling*

The the function `get_optimal_batch_size` is defined to dynamically determine the largest batch size that does not cause an out-of-memory (OOM) error on the GPU. This function creates a dummy batch of test sentences and iteratively increases the batch size until an OOM error is encountered. The batch size is then reduced by a factor of 2 and returned as the optimal batch size. This ensures efficient use of GPU memory

without manual tuning and guessing. Fig. 4.1 is a flowchart diagram, explaining the dynamic batch size determination process.

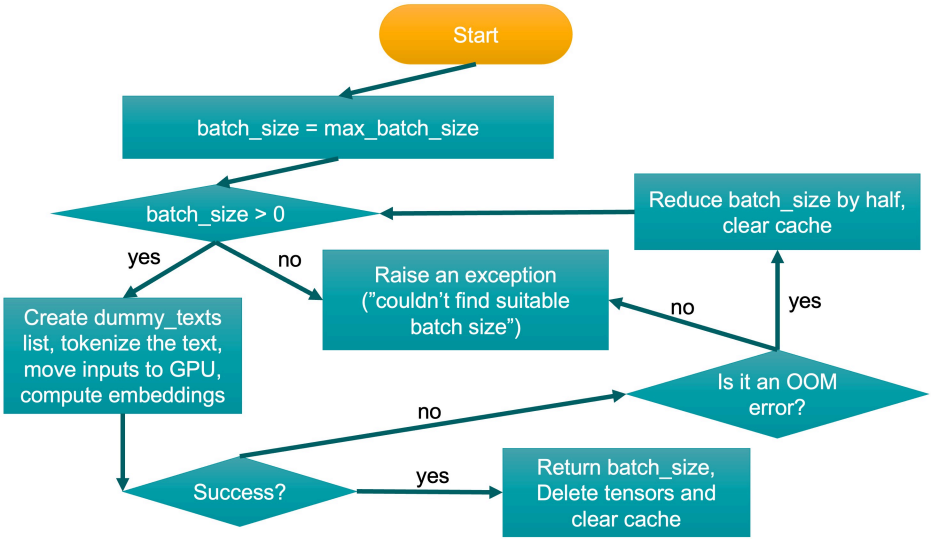



Figure 4.1: Flowchart, describing the funtion `get_optimal_batch_size` that is used to find the optimal batch size dynamically.

The function `mean_pooling` is defined to calculate the mean of the embeddings of the tokens in a sentence. This function is used to calculate the sentence embeddings for the dataset by aggregating the token embeddings into a single sentence embedding. It uses the attention mask to compute a weighted average (sum of embeddings divided by the sum of mask values) to generate a fixed-size embedding per input sentence.

### *4.3.4 Checkpointing and ChromaDB Initialization*

Since the HPC has a limited allowed runtime for each process (24h) the script had to be run multiple times to process the entire dataset for the selected languages. Therefore, it was necessary to come up with a way to resume the processing from where it was left off. A checkpoint loader function `load_checkpoint` was defined to load the last processed file and continue from there. This function loads a JSON file (checkpoint.json) that contains the last processed language to track which languages have already been processed to avoid redundant computation.

Next, a persistent ChromaDB client is initialized with a custom storage path inside the script direcory. It creates a collection (c4_embeddings) where the embeddings, along with other necessery metadata (like language information) will be stored.

To handle the existing data, a function `load_existing_data` is defined to query the ChromaDB collection for existing embeddings to determine how many samples have already been processed.

### *4.3.5 Processing the Dataset and Running the Embedding Model*

The main processing step starts here. Fig. 4.2 demonstrates all the necessary steps. For each language, the function `process_language` performs these steps:

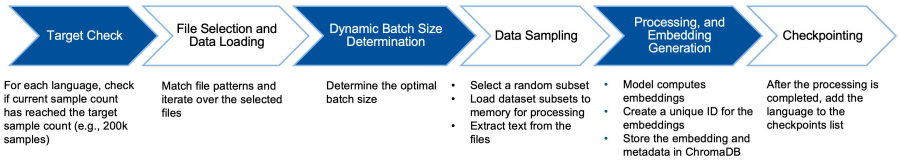



Figure 4.2: Steps of processing the dataset.

1. Target Check

   Compare the existing number of samples from the ChromaDB collection with the desired sample size (e.g., 200k) to determine if the language has already been processed. If the desired sample size has been reached, the language is skipped.

2. File Selection

   Uses `glob` to match file patterns (for english and multilingual files) and iterates over the files.

3. Dynamic Batch Size Determination

   Calls `get_optimal_batch_size` to determine the optimal batch size that can be used on the current GPU.

4. Data Sampling

   Randomly selects a subset of files (without any duplicates, using a `processed_files` set).

   Uses Hugging Face's streaming mode via `load_dataset` to efficiently load large JSON files without having to load the entire file into memory.

   Extracts the `text` field from each example until the needed samples are collected.

5. Embedding Data Generation

   The model computes embeddings, which are mean-pooled.

   The collected texts are batched and tokenized.

   A unique ID is generated for each sample using `uuid`.

   Embeddings and the language metadata and the original text are stored in the ChromaDB collection.

6. Checkpointing

   After the processing for a language is complete, the language code (two letters e.g., 'en') is stored in the checkpoint file.

Then the script loops through the list of defined languages and calls `process_language` for each language. The progress can be observed and tracked visually by the progress bars, and after completion, a final message confirms that all embeddings have been processed and stored.

# 5 Evaluation and Discussion

## 5.1 Required Scripts

To evaluate the generated embeddings, some additional scripts have been written. Most important ones are:

- `sampled_embeddings.py`: To sample 1k random points (vectors) from the generated embeddings for each language.
- `mean_embeddings.py`: To calculate the mean embeddings for each language.
- `tsne-mean-of-1k-samples.py`: To apply t-SNE on the mean embeddings.
- `interactive_tsne_plot.py`: To apply t-SNE on 1,000 random points and produce an interactive plot interactive plot with `plotly` that allows hovering over each data point to reveal its ID for retrieving the original sentence.
- `sampled_visualizer_gaussian.py`: To create a 2D Gaussian on top of the 1k random points for each language.
- `retrieve_sentence_by_id.py`: Where the input is the ID of the embedding, and it will retrieve the original sentence (before getting embedded).
- `cosine.py`: To compute the cosine similarity between the mean embeddings of each language.

Running these scripts yields a better understanding of the embeddings and the relationships between languages, as they produce plots and visualizations that are easier to interpret.

## 5.2 Research Question

This thesis investigates the types of clusters that emerge in the dataset. It examines whether the clustering is:

1. by linguistic similarities (shared language families) or
2. by thematic and topical similarities, where data points are grouped based on common subject areas (e.g., Science, Health, etc.).

To uncover the relationships between datapoints from different language families (represented by different colors) and determine whether they cluster based on their language groups, they first needed to be visualized. After visualizing a 24,000-point subset (1,000 per language) of the total of 4,800,000 embeddings generated (200,000 per language) and examining the t-SNE plot in Fig. 5.1, no clear clustering pattern was observed among datapoints belonging to the same language family (same color). To investigate further, the points overlapping in a randomly chosen region marked by the red arrow (in the lower-right section of the plot) were manually selected, retrieved, and translated to analyze their content.

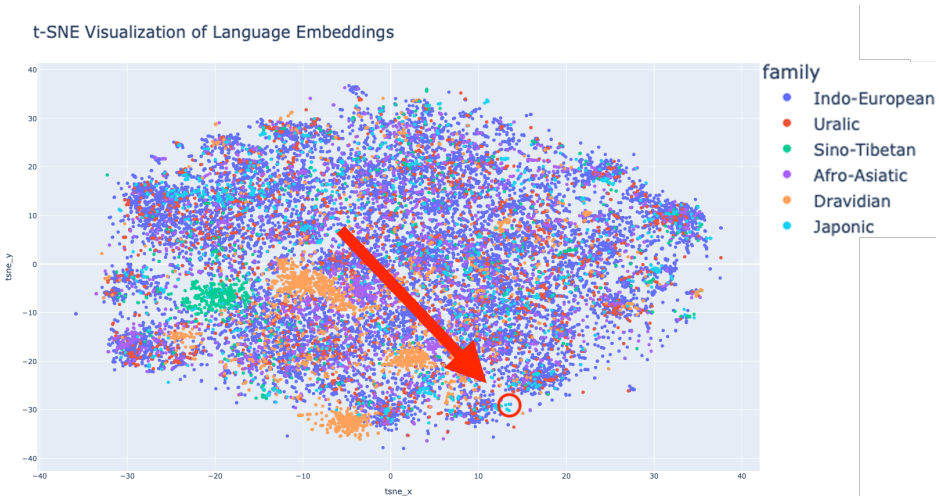
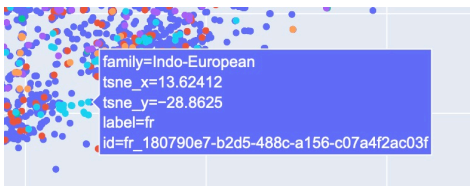
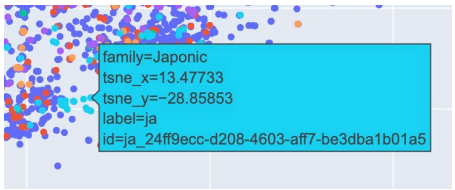



Figure 5.1: Random 24,000-point subset (1,000 per language) of the 4,800,000 total generated embeddings - Highlighted random section of the t-SNE plot in red, where points are overlapping

When zoomed in, two of the overlapping datapoints of different colors, randomly chosen for analysis, were related to French and Japanese text samples. It is guessed that the reason behind this overlap is a similarity between the topics or semantics of the original texts.
To test this, the original points, as seen in Fig. 5.2, were retrieved using `retrieve_sentence_by_id.py` and translated using DeepL:




((a)) French sample information




((b)) Japanese sample information

Figure 5.2: French and Japanese sample texts

The translations for each retrieved text:

> French datapoint: You live in Villenave-De-Rions and you are looking for a serious clairvoyant, recognized for her predictions and able to help you immediately in your next decisions. Contact me from Villenave-De-Rions Monday to Friday from 10am to 10pm and Saturday from 10am to 12pm. Villenave-De-Rions Live and online clairvoyance.

> Japanese datapoint: Feng Shui, Psychic, Psychic Reading, Clairvoyance, Spiritual Reading, Aura Reading, Aura Diagnosis, Chakras, Reiki Healing Love, Marriage, Work, Career, Luck, Life counseling. My name is Chris, a telephone fortune teller for women only. I use psychic reading to divine your future. By connecting with your guardian deity, I can help you with your life and future. I can read your purpose and your role in this life based on your past life evaluation. Chakra reading will purify your soul.

Both are advertisements with similar content, focusing on clairvoyance.

This observation suggests that the embedding model maps sentences with similar semantics to similar vectors. When these vectors are plotted using t-SNE, datapoints that appear close to each other represent text with related topics and meanings (sometimes so close that they may appear to strongly overlap). This behavior is observed regardless of the language (or the color of the datapoints). In other words, the multilingual embedding model used here encodes texts based on their semantics (the "gist" or "essence") rather than the specific language.

Up until this point, the influence of semantics on the positioning of the datapoints in the t-SNE plot has been tested. In the following analysis, the influence of languages and language families will be considered to determine whether they affect the positioning of the datapoints, or if the context and topics of the text are the only determining factors of the position of the datapoints on the t-SNE plot.

## 5.3 A Controlled Dataset for Understanding t-SNE Visualizations

To investigate further, a very small dataset was manually created, consisting of only 10 sentences across 10 distinct topics. These 10 sentences were manually translated into each of the 24 languages, producing a total of 240 sentences. The embedding model was run on this custom dataset, and t-SNE was applied to the resulting embeddings to visualize them in a shared space. This process aimed to deepen the understanding of the generated embeddings, with particular focus on the clusters formed and the relationships between the datapoints plotted using t-SNE.

Table 5.3 demonstrates the topics and sentences of the manually created dataset in English:

Table 5.3: Topics and their corresponding example sentences (10 total).

| **Topic** | **Sentence** |
|---|---|
| Sports | The match ended in a thrilling 2-2 draw. |
| Technology | Artificial intelligence is transforming industries worldwide. |
| News | A major earthquake struck the city, causing widespread damage. |
| Science | Astronomers discovered a new planet outside our solar system. |
| Health | Mental health awareness is crucial in today's fast paced world. |
| Environment | Climate change is one of the most pressing issues of our time. |
| Business | Stock markets around the world experienced significant growth last year. |
| Education | Online learning has become a popular alternative to traditional classrooms. |
| Literature | The poet captured the beauty of nature in vivid words. |
| Culture | Traditional festivals bring communities together in celebration. |

Fig. 5.4 shows the t-SNE plot of the 240 vectors (10 sentences × 24 languages).

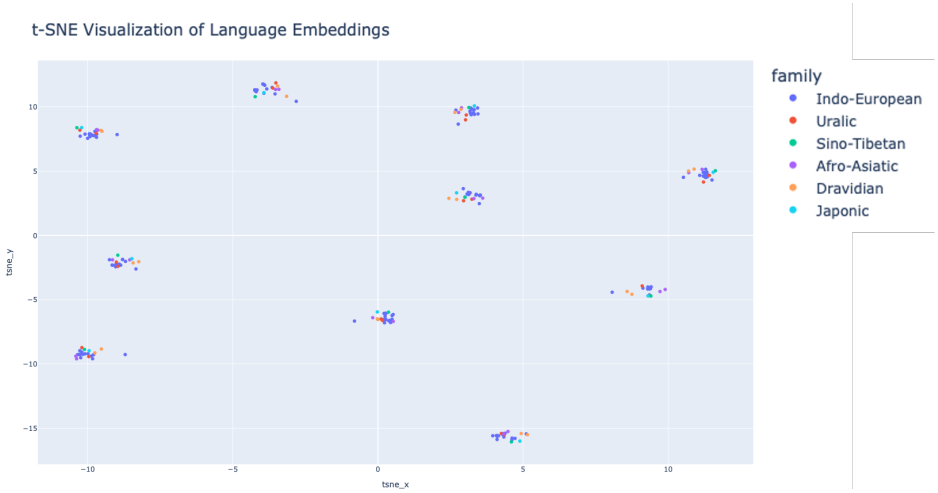



Figure 5.4: The t-SNE plot of 240 vectors (10 distinct sentences in 24 languages)

As can be observed, the embeddings visualized by the t-SNE plot appear to form 10 distinct clusters. The embedding model successfully captured the semantics of each sentence, indicating that, regardless of the language, the model discerns the semantic differences among the sentences. Since there were 10 distinct topics in the dataset, the embeddings appear in 10 separate clusters. This result is consistent with the hypothesis that the model encodes the semantics of the sentences, regardless of the language they are written in.

## 5.4 t-SNE Representation of Embeddings using Gaussian Fitting

Revisiting the previous plot with 1,000 points per language, to better visualize the structure of language embeddings and reduce the complexity of plotting all 1,000 individual points for each language resulting in 24,000 datapoints in total (as done in Fig. 5.1), in Fig. 5.5 each language was represented by a multivariate Gaussian distribution in the 2D t-SNE space, where the center corresponds to the mean embedding of the language, and the ellipse captures its spread and correlation structure. This approach provides a more interpretable visualization by summarizing the distributional properties of each language while highlighting areas of overlap and distinction between them.

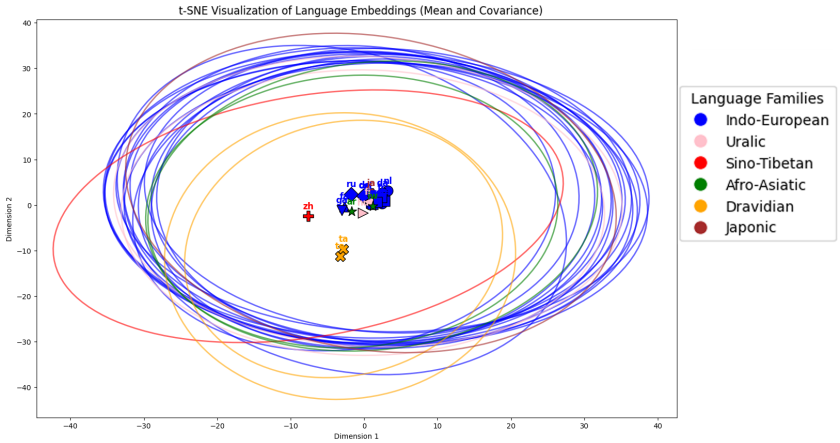



Figure 5.5: 1,000 randomly picked samples from each language (24 languages) after applying the 2D Gaussian

A strong overlap in the Gaussians is observed, indicating that the 1,000 points randomly selected for each language cover a similar set of topics. This behavior remains consistent across the chosen languages. Another important insight arises when zooming in on the mean embeddings and analyzing their behavior, which will be discussed in the next section.

To test whether the semantics of the topics in different languages are similar, the cosine similarity between the mean embeddings of each language was calculated. Fig. 5.6 shows the results:

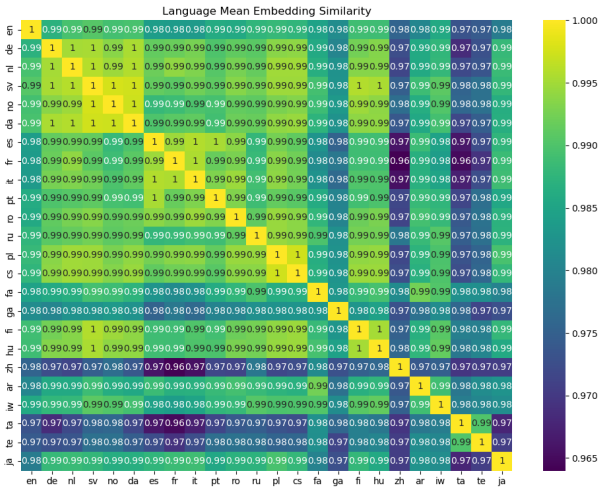



Figure 5.6: Cosine Similarity between the mean embeddings of each language

As it can be seen in Fig. 5.6, the cosine similarity for every pair of mean embeddings is consistently high (>0.9). However, this does not necessarily mean the languages themselves are '90% similar.' Rather, it reflects that most multilingual embedding spaces share a strong common component that dominates the angle between vectors [41]. It is important to note, that cosine similarity is a global measure, meaning it reduces each pair of vectors to a single number (the cosine of the angle). It does not consider any nonlinear structure or clusters, and it cannot gain knowledge of any groupings among multiple points. To be able to consider the structure of the embeddings, t-SNE was used instead of cosine similarity to visualize the embeddings in a 2D space.

There are two important insights here:

1. The Gaussians overlap strongly for all languages, indicating that the 1,000 points randomly selected for each language cover a diverse set of topics. This behavior is consistent across the chosen languages.
2. A more important and relevant insight arises when zooming in on the mean embeddings and analysing their behavior.

## 5.5 t-SNE Visualization of Mean Embeddings of Different Sample Sizes

To further investigate the clustering behavior of languages, their mean embeddings were analyzed. Computing the mean embedding over a large sample of sentences per language tends to reduce noise from topical differences and reveal underlying language-specific signals [42]. Prior work on multilingual models has similarly found that aggregating embeddings at the language level can highlight linguistic patterns more effectively than individual embeddings [43]. Accordingly, in this thesis, the mean embedding for each language was used to examine whether languages from the same family tend to cluster together.

The means of the 10 embeddings (per language) from Fig. 5.4 were taken to represent each language with a single vector. These were then plotted in Fig. 5.7, with language families marked to observe whether any meaningful clusters emerged.

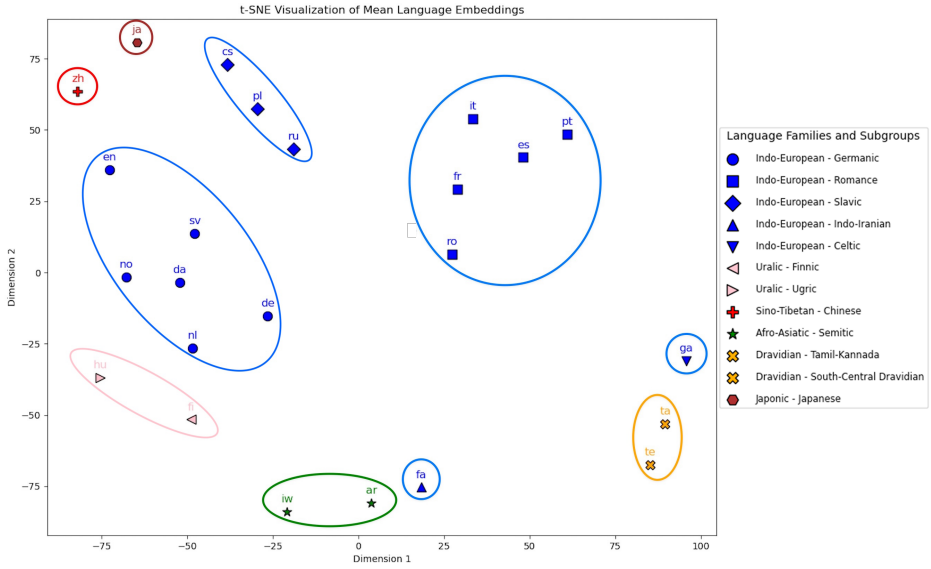



Figure 5.7: Clusters of languages based on their mean embeddings from 10 samples

The same process has been repeated in Fig. 5.8 for 1,000 samples per language:

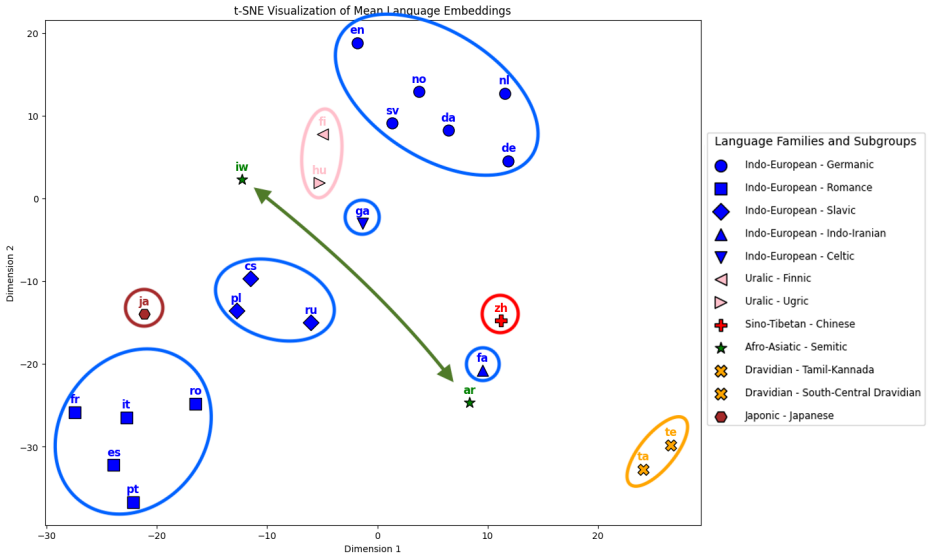



Figure 5.8: Clusters of languages based on their mean embeddings from 1,000 samples per language, grouped by language family

Examining the plot reveals that languages from the same families and subgroups tend to cluster near each other, as seen in Fig. 5.8. This raises the question of whether the clustering is coincidental or if languages from the same families consistently cluster together when their mean embeddings are analyzed.

Thus far, two sample sizes have been tested:

1. 10 samples per language
2. 1,000 samples per language

In both scenarios, the relationships between languages were consistent, forming clusters of languages from the same families.

However, only two sample sizes (especially small ones) may not be sufficient.

Accordingly, this process was extended to larger sample sizes. The same procedure was repeated for datasets containing:

- 10,000 samples per language
- 20,000 samples per language
- 50,000 samples per language
- 100,000 samples per language
- 200,000 samples per language

The goal was to determine whether clustering behavior based on language families remains consistent as the sample size increases.

Below are the results for these sample sizes:

**10,000 samples per language:**

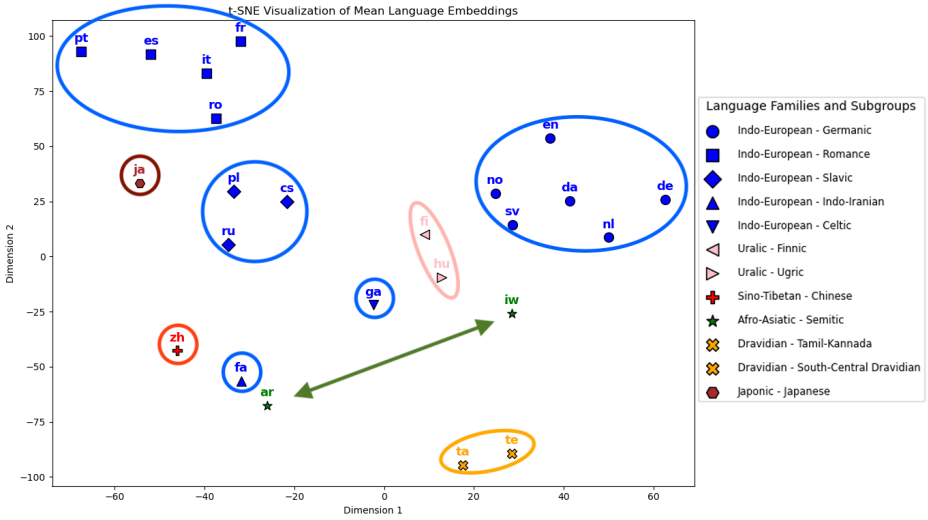



Figure 5.9: t-SNE clusters for 10,000 samples per language

**20,000 samples per language:**

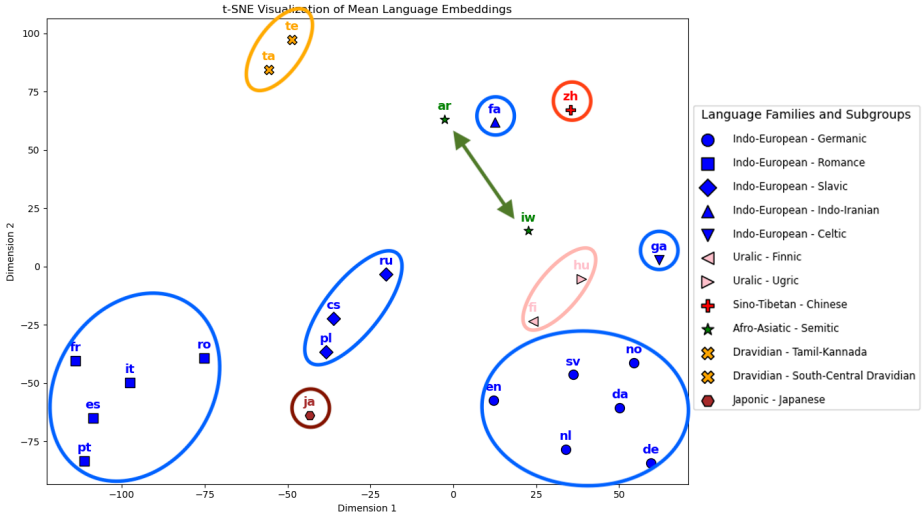



Figure 5.10: t-SNE clusters for 20,000 samples per language

**50,000 samples per language:**

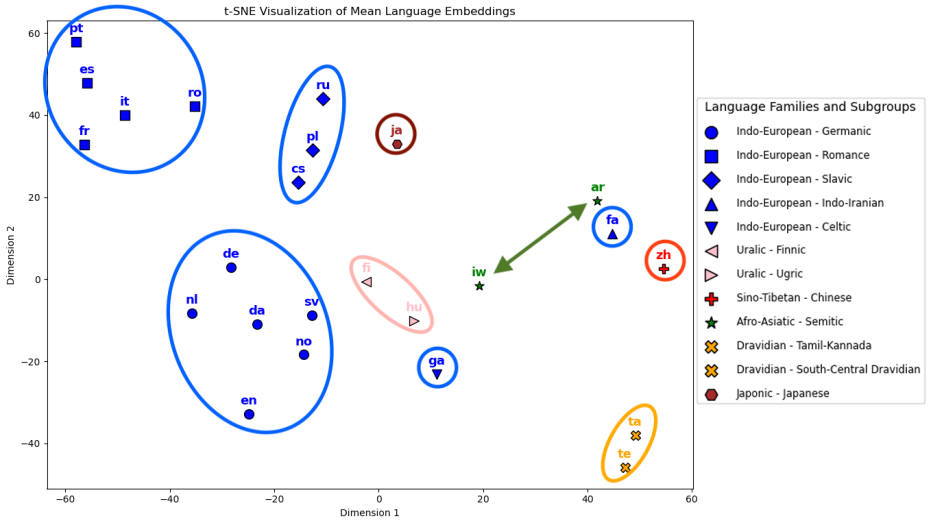



Figure 5.11: t-SNE clusters for 50,000 samples per language

**100,000 samples per language:**

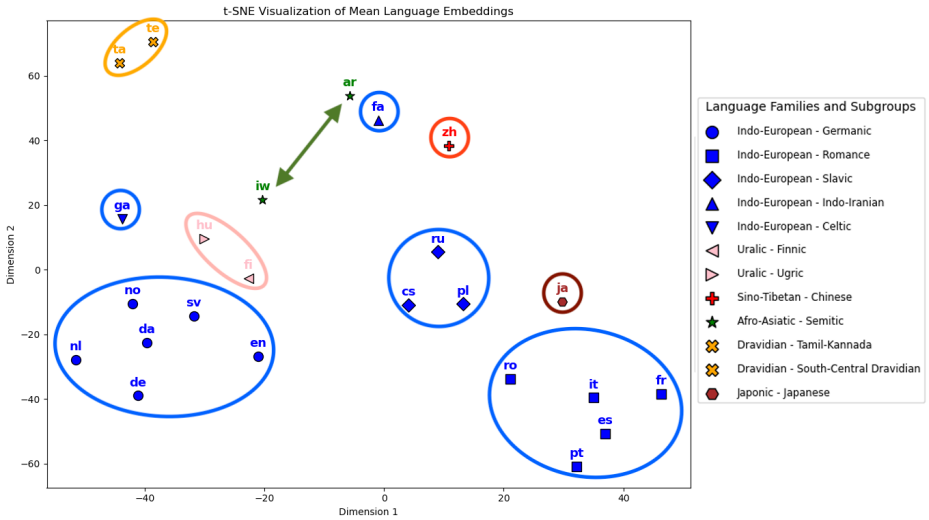



Figure 5.12: t-SNE clusters for 100,000 samples per language

**200,000 samples per language:**

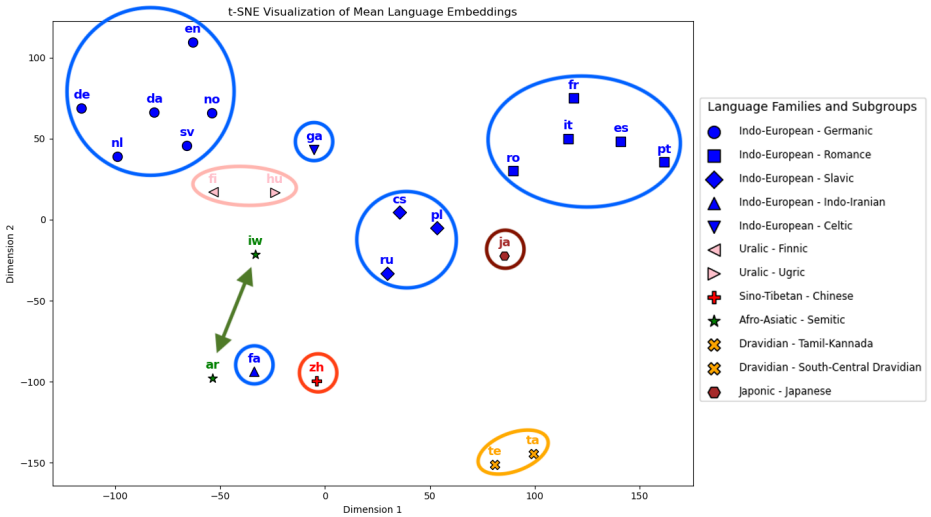



Figure 5.13: t-SNE clusters for 200,000 samples per language

## 5.6 Lessons Learned

As shown in the figures, the relationships between languages remain consistent across all sample sizes. Clusters of languages from the same family persist and maintain their expected arrangements. The t-SNE plots suggest that languages from the same family occupy nearby regions in the embedding space, indicating that the model encodes a partial reflection of linguistic proximity. Similar findings have been reported, for instance, [44] showed that word embedding spaces can reconstruct language phylogenies.

Therefore, even though the embedding model is based on textual semantics, an entire language can be represented by the mean of its embeddings. Comparing these mean vectors enables the assessment of inter-language relationships.

This hypothesis was tested with the following sample sizes:

1. 10 samples per language
2. 1,000 samples per language
3. 10,000 samples per language
4. 20,000 samples per language
5. 50,000 samples per language
6. 100,000 samples per language
7. 200,000 samples per language

Through this process, it was demonstrated that two types of clusters can be identified by applying the multilingual embedding model to the multilingual dataset:

1. Languages from similar families cluster together when analysing the mean embeddings.
2. Texts on similar topics cluster together (e.g., sports-related data, regardless of language).

In summary, both effects can be observed, depending on how the embeddings are aggregated and analyzed.

If one examines mean embeddings, insights about the languages themselves and their families emerge.

If one examines the raw embeddings, insights about the dataset topics become evident.

# 6 Conclusion

In this thesis, a systematic analysis was conducted to uncover the relations and proximity of multilingual texts in a shared embedding space, with particular emphasis on linguistic clustering versus semantic clustering. Using the BAAI bge-m3 embedding model and a vector database infrastructure, hundreds of thousands of sentences from 24 languages were processed to form a high-dimensional representation. By employing t-SNE to reduce the dimensionality for visualization, it became evident that thematically similar texts cluster closely regardless of their source language. At the same time, certain language families still present recognizable clusters when analysing the mean embeddings, supporting the notion that the embedding model captures some linguistic features alongside topical similarities.

One key element of the deployment was the inclusion of parallel processing on a high-performance computing (HPC) architecture, which accelerated overall computation and allowed larger scope exploration into the data. One of the main challenges working with massive multilingual corpora was using the available resources efficiently and effectively without reaching bottlenecks like out-of-memory (OOM) errors. This demonstrates that High-Performance Computing resources can help make important experiments possible, especially when dealing with extremely large multilingual corpora. When it comes to the clustering and grouping of languages, the results of this thesis aligns with conventional understandings of language families as established through geographical, historical, and linguistic research. This consistency suggests the validity of the analysis and the accuracy of the employed methodology in capturing linguistic features and relationships between these languages.

The results of the thesis suggest that if topical similarity and analysis are the primary goals, the raw multilingual embeddings can be used directly. However, if the focus is on linguistic clustering, to uncover insights about language relations and distances, it is beneficial to use the mean embeddings of the languages.

# References


[1] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. "Machine Learning Testing: Survey, Landscapes and Horizons". In: *CoRR* abs/1906.10742 (2019). arXiv: 1906.10742.

[2] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67.

[3] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz. "Machine Learning With Big Data: Challenges and Approaches". In: *IEEE Access* 5 (2017), pp. 7776–7797.

[4] T. A. Chang, C. Arnett, Z. Tu, and B. K. Bergen. *When Is Multilinguality a Curse? Language Modeling for 250 High- and Low-Resource Languages*. 2023. arXiv: 2311.09205 [cs.CL].

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[6] J. Pennington, R. Socher, and C. D. Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[7] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606* (2016).

[8] J. Allen. *Natural Language Understanding*. 2nd. Benjamin/Cummings, 1994.

[9] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. 4th. Morgan Kaufmann, 2016.

[10] D. Nadeau and S. Sekine. "A survey of named entity recognition and classification". In: *Lingvisticae Investigationes* 30.1 (2007), pp. 3–26.

[11] Z. Harris. "Distributional Structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[12] H. P. Luhn. "The Automatic Creation of Literature Abstracts". In: *IBM Journal of Research and Development* 2.2 (1958), pp. 159–165.

[13] J. Sinclair. *Corpus, Concordance, Collocation*. Oxford University Press, 1991.

[14] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2 (1993), pp. 313–330.

[15] T. Winograd. *Language as a Cognitive Process: Syntax*. Addison-Wesley, 1983.

[16] K. Schuster and M. Nakajima. *Japanese and Korean Tokenization Rules*. https://ai.googleblog.com/2012/09/japanese-and-korean-tokenization-rules.html. Accessed: 2024-01-14. 2012.

[17] M. F. Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137.

[18] A. Mikheev. "Automatic rule induction for unknown-word guessing". In: *Computational Linguistics* 23.3 (1997), pp. 405–423.

[19] G. E. Hinton and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[20] R. P. Adams, H. Wallach, and Z. Ghahramani. "Learning the structure in latent spaces for language modeling". In: *Advances in Neural Information Processing Systems*. Vol. 24. 2011.

[21] G. Salton and C. Buckley. *Term-weighting Approaches in Automatic Text Retrieval*. Addison-Wesley, 1975.

[22] J. Johnson, M. Douze, and H. Jégou. "Billion-scale Similarity Search with GPUs". In: *IEEE Transactions on Big Data*. FAISS: widely used for vector similarity search. 2019.

[23] J. Ba, V. Mnih, and K. Kavukcuoglu. "Multiple Object Recognition with Visual Attention". In: *arXiv preprint* (2014). arXiv 1412.7755.

[24] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv preprint* (2015). arXiv 10.48550/ARXIV.1409.0473.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *Google Brain, Google Research, University of Toronto* (2017).

[26] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals. "Listen, Attend and Spell". In: *arXiv preprint* (2015). arXiv 10.48550/ARXIV.1508.01211.

[27] P. Poupart. "cs480-lecture19-slides". In: *University of Waterloo Canada* (2019).

[28] G. B. Team. *Tensor2Tensor Intro*. ONLINE, accessed 20.Feb.2023. 2018.

[29] W. Jiang, X. Li, H. Hu, Q. Lu, and B. Liu. "Multi-Gate Attention Network for Image Captioning". In: *IEEE Access* PP (2021). 10.1109/ACCESS.2021.3067607, pp. 1–1.

[30] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.9 (2008).

[31] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer". In: *arXiv preprint arXiv:2010.11934* (2020).

[32] A. I. for AI. *C4 Multilingual Dataset*. Accessed: 2024-01-14. 2021.

[33] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2024. arXiv: 2402.03216 [cs.CL].

[34] Ç. Yıldız, N. K. Ravichandran, P. Punia, M. Bethge, and B. Ermis. "Investigating Continual Pretraining in Large Language Models: Insights and Implications". In: *arXiv preprint* (2024). arXiv:2402.17400v1.

[35] M. Reid, M. T. Jha, C. Freer, C. Raffel, and G. Melis. "M2D2: A Massively Multi-Domain Dataset for Better Domain Adaptation and Transfer Learning". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2022.

[36] N. Reimers and I. Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics. 2019, pp. 3982–3992.

[37] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020, pp. 8342–8360.

[38] A. Cossu, D. Belouadi, V. Lomonaco, D. Maltoni, A. Popescu, E. Bechara, and E. Hadoux. "Continual learning for Natural Language Processing: A survey". In: *Proceedings of the Neural Information Processing Systems (NeurIPS)* (2022).

[39] X. Ke, L. Sun, W. Zhao, and M. Xu. "Soft-Masking for Continual Pretraining: Balancing Adaptation and Generalization". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2023.

[40] H. Face. *MTEB Leaderboard*. https://huggingface.co/spaces/mteb/leaderboard. Accessed: 2025-01-10.

[41] J. Mu and P. Viswanath. "All-but-the-Top: Simple and Effective Postprocessing for Word Representations". In: *International Conference on Learning Representations*. 2018.

[42] S. Wu and M. Dredze. "Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 833–844.

[43] J. Libovický, A. Fraser, and R. Rosa. "Language Model Analysis in Cross-Lingual Transfer". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 7693–7707.

[44] O. Draganov and S. Skiena. "The Shape of Word Embeddings: Quantifying Non-Isometry with Topological Data Analysis". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 12080–12099.

# Appendix

## Appendix A

Specifications of the used HPC clusters:

**JUWELS Booster**

- **Compute Nodes:** 936 nodes
- **CPU:** 2× AMD EPYC Rome 7402, each with 24 cores at 2.8 GHz
- **Memory:** 512 GB DDR4 at 3200 MHz
- **GPUs:** 4× NVIDIA A100 per node, each with 40 GB HBM2e
- **Network:** 4× InfiniBand HDR (Connect-X6)
- **Peak Performance:** 73 Petaflops

**JURECA DC (Phase 2)**

- **Standard Compute Nodes:** 480 nodes
  - **CPU:** 2× AMD EPYC 7742, each with 64 cores at 2.25 GHz
  - **Memory:** 512 (16 x 32) GB DDR4 at 3200 MHz
  - **Network:** InfiniBand HDR100 (NVIDIA Mellanox Connect-X6)
- **Large-Memory Compute Nodes:** 96 nodes
  - **CPU:** 2× AMD EPYC 7742, each with 64 cores at 2.25 GHz
  - **Memory:** 1024 (16 x 64) GB DDR4 at 3200 MHz
  - **Network:** InfiniBand HDR100 (NVIDIA Mellanox Connect-X6)
- **Accelerated Compute Nodes:** 192 nodes
  - **CPU:** 2× AMD EPYC 7742, each with 64 cores at 2.25 GHz
  - **Memory:** 512 (16 x 32) GB DDR4 at 3200 MHz
  - **GPUs:** 4× NVIDIA A100 per node, each with 40 GB HBM2e
  - **Network:** 2× InfiniBand HDR (NVIDIA Mellanox Connect-X6)
- **Login Nodes:** 12 nodes
  - **CPU:** 2× AMD EPYC 7742, each with 64 cores at 2.25 GHz
  - **Memory:** 1024 (16 x 64) GB DDR4 at 3200 MHz
  - **GPUs:** 2× NVIDIA Quadro RTX8000

  - **Network:** InfiniBand HDR100 (NVIDIA Mellanox ConnectX6) and 100 Gigabit Ethernet external connection
- **Performance:** 3.54 Petaflops (CPU) + 14.98 Petaflops (GPU) peak performance
- **Total CPU Cores:** 98,304
- **Total GPUs:** 768
- **Network:** Mellanox InfiniBand HDR (HDR100/HDR) DragonFly+ network with approximately 15 Tb/s connection to Booster via gateway nodes
- **Storage Connection:** 350 GB/s network connection to JUST for storage access

# Appendix B

Code Snippets:

parallelprocessing.py

```
import os
import glob
import json
import sys
import uuid
import gc
import random
import torch
from tqdm import tqdm
from transformers import AutoTokenizer, AutoModel
from datasets import load_dataset
import chromadb

# Suppress tokenizer parallelism warnings
os.environ["TOKENIZERS_PARALLELISM"] = "false"

# Ensure offline mode for datasets if needed
os.environ['HF_DATASETS_OFFLINE'] = '1'

# Set the datasets cache directory to a location with sufficient
    space
os.environ['HF_DATASETS_CACHE'] =
    '/p/scratch/atmlaml/rahmdel1/datasets_cache'

# Base directory where the dataset is stored
data_dir = "/p/scratch/atmlaml/rahmdel1/Dataset_mc4/c4"

# Specify the directories to include
subdirs = ["en", "multilingual"]

# Languages of interest
languages = [
   "en", "de", "nl", "sv", "no", "da", "es", "fr", "it", "pt",
```

```
    "ro", "ru", "pl", "cs", "fa", "ga", "fi", "hu", "zh", "ar",
    "iw", "ta", "te", "ja"
]

# Define sequence length and initial maximum batch size
model_name = 'BAAI/bge-m3' # Use your desired model
SEQLEN = 512 # Adjusted max sequence length in tokens
MAX_BATCH_SIZE = 2048 # Adjust based on your GPU capacity

# Variable to set sample size per language
sample_size_per_language = 200000 # Adjust this value as needed

# Checkpoint file path
checkpoint_file = 'checkpoint.json'

# Initialize tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name).to('cuda')
model.eval()

# Function to get optimal batch size
def get_optimal_batch_size(seqlen, tokenizer, model,
    max_batch_size):
    batch_size = max_batch_size
    while batch_size > 0:
        try:
            dummy_texts = ['This is a test sentence.'] * batch_size
            inputs = tokenizer(
                dummy_texts,
                return_tensors='pt',
                padding='max_length',
                truncation=True,
                max_length=seqlen,
            )
            inputs = {key: value.to('cuda') for key, value in
                inputs.items()}
            with torch.no_grad():
                outputs = model(**inputs)
                embeddings = mean_pooling(outputs,
                    inputs['attention_mask'])
            del inputs, outputs, embeddings
            torch.cuda.empty_cache()
            gc.collect()
            return batch_size
        except RuntimeError as e:
            if 'out of memory' in str(e).lower():
                print(f"Batch size {batch_size} too large,
                    reducing...")
                batch_size = batch_size // 2
                torch.cuda.empty_cache()
                gc.collect()
            else:
                raise e
    raise Exception('Could not find suitable batch size.')
```

```
    # Define mean pooling function
    def mean_pooling(model_output, attention_mask):
        token_embeddings = model_output.last_hidden_state # First
            element contains token embeddings
        input_mask_expanded = attention_mask.unsqueeze(-1)
        .expand(token_embeddings.size()).float()
        sum_embeddings = torch.sum(token_embeddings *
            input_mask_expanded, dim=1)
        sum_mask = input_mask_expanded.sum(dim=1)
        embeddings = sum_embeddings / sum_mask.clamp(min=1e-9)
        return embeddings

    # Function to load checkpoint
    def load_checkpoint():
        if os.path.exists(checkpoint_file):
            with open(checkpoint_file, 'r') as f:
                content = f.read()
                if content.strip():
                    checkpoint_data = json.loads(content)
                    print(f"Loaded checkpoint data.")
                else:
                    checkpoint_data = []
                    print("Checkpoint file is empty. Starting fresh.")
        else:
            checkpoint_data = []
        return checkpoint_data

    # Initialize Chroma client with the custom storage directory
    client = chromadb.PersistentClient(
        path="/p/scratch/atmlaml/rahmdel1/sourcecode/200k/
        chroma-embeddings")

    # Create or get a collection in Chroma
    collection_name = "c4_embeddings"
    collection = client.get_or_create_collection(name=collection_name)

    # Load checkpoint data
    processed_languages = load_checkpoint()

    # Function to get the existing sample count for a language
    def get_existing_sample_count(lang):
        batch_size = 10000 # Adjust as needed
        offset = 0
        total_ids = 0

        while True:
            results = collection.get(
                ids=None,
                where={"lang": lang},
                include=[], # Exclude embeddings, metadatas, documents
                    for efficiency
                limit=batch_size,
                offset=offset
            )
            batch_ids = results['ids']
```

```
            num_ids = len(batch_ids)
            total_ids += num_ids

            if num_ids < batch_size:
                break # No more data to fetch
            offset += batch_size

        return total_ids

    # Function to process a single language
    def process_language(lang):
        total_samples_needed = sample_size_per_language

        # Get existing sample count
        existing_sample_count = get_existing_sample_count(lang)
        print(f"Existing samples for {lang}: {existing_sample_count}")

        # If the language has already been processed to the desired
            sample size, skip it
        if existing_sample_count >= total_samples_needed:
            print(f"Language {lang} already has
                {existing_sample_count} samples. Skipping.")
            # Update checkpoint if not already updated
            if lang not in processed_languages:
                processed_languages.append(lang)
                with open(checkpoint_file, 'w') as f:
                    json.dump(processed_languages, f)
                print(f"Updated checkpoint with language: {lang}")
            return

        print(f"Processing language: {lang}")

        # Collect all files for the language
        data_files = []
        for subdir in subdirs:
            subdir_path = os.path.join(data_dir, subdir)
            if lang == "en" and subdir == "en":
                data_files_pattern = f"{subdir_path}/c4-train.*.json.gz"
            elif lang != "en" and subdir == "multilingual":
                data_files_pattern =
                    f"{subdir_path}/c4-{lang}.*.json.gz"
            else:
                continue
            data_files.extend(sorted(glob.glob(data_files_pattern)))

        if not data_files:
            print(f"No files found for language {lang}")
            return

        # Determine optimal batch size
        batch_size = get_optimal_batch_size(SEQLEN, tokenizer, model,
            max_batch_size=MAX_BATCH_SIZE)
        print(f"Using batch size: {batch_size}")

        # Keep track of total samples collected in this run
```

```
        total_new_samples_collected = 0

        # Keep a list of files already processed to avoid duplicates
        processed_files = set()

        # While we need more samples
        while existing_sample_count + total_new_samples_collected <
            total_samples_needed:
            samples_needed = total_samples_needed -
                (existing_sample_count + total_new_samples_collected)
            print(f"Samples needed for {lang}: {samples_needed}")

            # Randomly select files that haven't been processed yet
            available_files = list(set(data_files) - processed_files)
            if not available_files:
                print(f"No more files available for language {lang}.
                    Cannot reach target sample size.")
                break

            num_files_to_select = min(5, len(available_files)) #
                Adjust the number of files to select as needed
            selected_files = random.sample(available_files,
                num_files_to_select)
            processed_files.update(selected_files)
            print(f"Selected {num_files_to_select} new random files
                for language {lang}")

            # Collect sampled examples
            sampled_examples = []
            total_examples_collected = 0

            for file_path in selected_files:
                # Load dataset from the file in streaming mode
                dataset = load_dataset(
                    'json',
                    data_files=file_path,
                    split='train',
                    streaming=True
                )

                for example in dataset:
                    text = example.get('text', '')
                    if text:
                        sampled_examples.append(text)
                        total_examples_collected += 1
                    if total_examples_collected >= samples_needed:
                        break
                if total_examples_collected >= samples_needed:
                    break

            total_samples = len(sampled_examples)
            print(f"Collected {total_samples} new samples for {lang}")

            # Process sampled examples in batches
            for i in tqdm(range(0, total_samples, batch_size),
```

```
                desc=f"Processing {lang}"):
            batch_texts = sampled_examples[i:i+batch_size]
            inputs = tokenizer(
                batch_texts,
                return_tensors='pt',
                padding='max_length',
                truncation=True,
                max_length=SEQLEN,
            ).to('cuda')

            with torch.no_grad():
                outputs = model(**inputs)
                embeddings = mean_pooling(outputs,
                    inputs['attention_mask'])

            # Prepare data for ChromaDB
            batch_ids = [f"{lang}_{uuid.uuid4()}" for _ in
                range(len(batch_texts))]
            batch_metadatas = [{'lang': lang} for _ in
                range(len(batch_texts))]

            # Save embeddings to ChromaDB
            collection.add(
                embeddings=embeddings.cpu().numpy().tolist(),
                ids=batch_ids,
                metadatas=batch_metadatas,
                documents=batch_texts
            )

            # Clear variables
            del inputs, outputs, embeddings
            torch.cuda.empty_cache()
            gc.collect()

        total_new_samples_collected += total_samples
        print(f"Total new samples collected for {lang}:
            {total_new_samples_collected}")

        # If no new samples were collected, break to avoid
            infinite loop
        if total_samples == 0:
            print(f"No new samples collected for {lang}. Cannot
                reach target sample size.")
            break

    final_sample_count = existing_sample_count +
        total_new_samples_collected
    print(f"Final sample count for {lang}: {final_sample_count}")

    if final_sample_count >= total_samples_needed:
        print(f"Reached target sample size for {lang}.")
        # Update checkpoint
        processed_languages.append(lang)
        with open(checkpoint_file, 'w') as f:
            json.dump(processed_languages, f)
```

```
            print(f"Updated checkpoint with language: {lang}")
        else:
            print(f"Could not reach target sample size for {lang}.
                Final sample count: {final_sample_count}")
            # Optionally update checkpoint to avoid reprocessing
            processed_languages.append(lang)
            with open(checkpoint_file, 'w') as f:
                json.dump(processed_languages, f)
            print(f"Updated checkpoint with language: {lang} (partial
                data)")

    # Process each language
    for lang in languages:
        process_language(lang)

    print("All embeddings have been processed and stored in Chroma.")
```

# License